

DRSWAD

Light OJ DP Problems

With Solutions

By DrSwad

2018

NAYEEMUL ISLAM SWAD

1004 - Monkey Banana Problem

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are in the world of mathematics to solve the great "Monkey Banana Problem". It states that, a monkey enters into a diamond shaped two dimensional array and can jump in any of the adjacent cells **down** from its current position (see figure). While moving from one cell to another, the monkey eats all the bananas kept in that cell. The monkey enters into the array from the upper part and goes out through the lower part. Find the maximum number of bananas the monkey can eat.

Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Every case starts with an integer N ($1 \leq N \leq 100$). It denotes that, there will be $2*N - 1$ rows. The i^{th} ($1 \leq i \leq N$) line of next N lines contains exactly i numbers. Then there will be $N - 1$ lines. The j^{th} ($1 \leq j < N$) line contains $N - j$ integers. Each number is greater than zero and less than 2^{15} .

Output

For each case, print the case number and maximum number of bananas eaten by the monkey.

Sample Input	Output for Sample Input
2 4 7 6 4 2 5 10 9 8 12 2 2 12 7 8 2 10 2 1 2 3 1	Case 1: 63 Case 2: 5

Note

Dataset is huge, use faster I/O methods.

SOLUTION

```
#include <iostream>
#include <string.h>
#include <fstream>
using namespace std;
int main()
{
    int t;
    int jmax;
    int n;
    //fstream cin;
    //cin.open("1004.in", ios::in);
    unsigned long long a[300][300];
    unsigned long long inp[300][300];
    n = 4;
    cin >> t;
    for (int k = 1; k <= t; k++) {

        cin >> n;
        memset(a, 0, sizeof(a));
        memset(inp, 0, sizeof(inp));

        for (int i = 0; i <= ((2 * n) - 1); i++) {
            if(i < n)
                jmax = i;
            else
                jmax = 2 * n - i;

            for(int j = 0; j < jmax; j++) {
                cin >> inp[i][j];
            }
        }

        for (int i = ((2 * n) - 1); i >= 0; i--) {

            if(i < n) {
                jmax = i+1;
            }

            else {
                jmax = 2 * n - i;
            }

            if(i < n) {
```

```

        for(int j = 0; j < jmax; j++) {
            if(i == ((2 * n) - 1)) {
                a[i][j] = inp[i][j];
                continue;
            }

            a[i][j] = max((inp[i][j] + a[i+1][j]),
(inp[i][j] + a[i+1][j+1]));
        }
    }
    else {
        for (int j = jmax - 1; j >= 0; j--) {
            if(i == ((2 * n) - 1)) {
                a[i][j] = inp[i][j];
                continue;
            }

            if(j - 1 >= 0) {
                a[i][j] =
max((inp[i][j]+a[i+1][j]), (inp[i][j]+ a[i+1][j-1]));
            }
            else {
                a[i][j] = a[i+1][j] + inp[i][j];
            }
        }
    }
}
cout <<"Case "<<k<<": "<< a[0][0]<<endl;
}
}

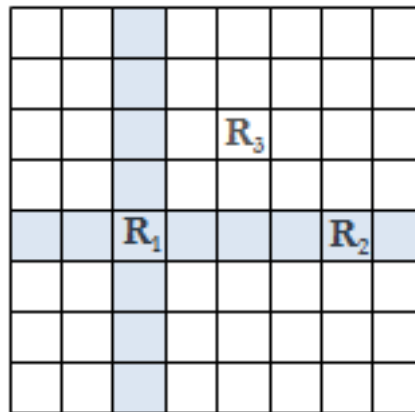
```

1005 - Rooks

Time Limit: 1 second(s)

Memory Limit: 32 MB

A rook is a piece used in the game of chess which is played on a board of square grids. A rook can only move vertically or horizontally from its current position and two rooks attack each other if one is on the path of the other. In the following figure, the dark squares represent the reachable locations for rook R_1 from its current position. The figure also shows that the rook R_1 and R_2 are in attacking positions where R_1 and R_3 are not. R_2 and R_3 are also in non-attacking positions.



Now, given two numbers n and k , your job is to determine the number of ways one can put k rooks on an $n \times n$ chessboard so that no two of them are in attacking positions.

Input

Input starts with an integer T (≤ 350), denoting the number of test cases.

Each case contains two integers n ($1 \leq n \leq 30$) and k ($0 \leq k \leq n^2$).

Output

For each case, print the case number and total number of ways one can put the given number of rooks on a chessboard of the given size so that no two of them are in attacking positions. You may safely assume that this number will be less than 10^{17} .

Sample Input	Output for Sample Input
8	Case 1: 1
1 1	Case 2: 4
2 1	Case 3: 9
3 1	Case 4: 16
4 1	Case 5: 72
4 2	Case 6: 96
4 3	Case 7: 24
4 4	Case 8: 0
4 5	

SOLUTION

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <cmath>
#include <algorithm>
#include <list>
#include <stack>
#include <utility>
#include <ctime>
#include <string>
#include <map>
#define LongInt long long
#define SIZE 1000
#define max3(a, b, c) max(a, b) > max(b, c) ? max(a, b) : max(b, c)
#define min3(a, b, c) min(a, b) < min(b, c) ? min(a, b) : min(b, c)
#define digit(c) (c - '0')
using namespace std;
int a[SIZE][SIZE];
unsigned LongInt solve(LongInt n, LongInt m, LongInt k)
{
    if(k > n || k > m)
        return 0;
    else if(k == 0)
        return 1;
    else if(m == 1)
        return n;
    else return n * solve(n - 1, m - 1, k - 1) + solve(n, m - 1, k);
}
int main()
{
    //freopen("input.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
    int tc, k, n;
    unsigned LongInt l;

    scanf("%d", &tc);

    for(int p = 0; p < tc; p++)
    {
        scanf("%d %d", &n, &k);
        //for(int i = 0; i <
```

```
        printf("Case %d: %llu\n", p + 1, solve(n, n, k));  
    }  
  
    return 0;  
}
```


1011 - Marriage Ceremonies

Time Limit: 2 second(s)

Memory Limit: 32 MB

You work in a company which organizes marriages. Marriages are not that easy to be made, so, the job is quite hard for you.

The job gets more difficult when people come here and give their bio-data with their preference about opposite gender. Some give priorities to family background, some give priorities to education, etc.

Now your company is in a danger and you want to save your company from this financial crisis by arranging as much marriages as possible. So, you collect N bio-data of men and N bio-data of women. After analyzing quite a lot you calculated the priority index of each pair of men and women.

Finally you want to arrange N marriage ceremonies, such that the total priority index is maximized. Remember that each man should be paired with a woman and only monogamous families should be formed.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case contains an integer N ($1 \leq n \leq 16$), denoting the number of men or women. Each of the next N lines will contain N integers each. The j^{th} integer in the i^{th} line denotes the priority index between the i^{th} man and j^{th} woman. All the integers will be positive and not greater than 10000.

Output

For each case, print the case number and the maximum possible priority index after all the marriages have been arranged.

Sample Input	Output for Sample Input
2 2 1 5 2 1 3 1 2 3 6 5 4 8 1 2	Case 1: 7 Case 2: 16

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <map>
#include <utility>
#define check(n, pos) (n & (1<<pos))
#define set(n, pos) (n | (1<<pos))
using namespace std;
int n;
long long a[20][20];
int mm[17][1<<17];
long long wow(int i, long long vis)
{
    long long sum;
    sum = -1;
    if(i == n) {
        return 0;
    }
    if(mm[i][vis] == -1) {
        for (int j = 0; j < n; j++) {
            if(check(vis, j) == 0) {
                sum = max(sum, a[i][j] + wow(i+1, set(vis,
j))));
            }
        }
        mm[i][vis] = sum;
        return sum;
    }
    else {
        return mm[i][vis];
    }
}

int main()
{
    int t;
    int x;
    long long sum;
    long long vis;
    vis = 0;
    scanf("%d", &t);
    for (int i = 1; i <= t; i++) {
        scanf("%d", &n);
```

```
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                scanf("%lld", &a[j][k]);
            }
        }

        memset(mm, -1, sizeof(mm));
        vis = 0;
        sum = wow(0, vis);
        printf("Case %d: %lld\n", i, sum);
    }
}
```

1013 - Love Calculator

Time Limit: 2 second(s)

Memory Limit: 32 MB

Yes, you are developing a '**Love calculator**'. The software would be quite complex such that nobody could crack the exact behavior of the software.

So, given two names your software will generate the percentage of their '**love**' according to their names. The software requires the following things:

1. The length of the shortest string that contains the names as subsequence.
2. Total number of unique shortest strings which contain the names as subsequence.

Now your task is to find these parts.

Input

Input starts with an integer **T** (≤ 125), denoting the number of test cases.

Each of the test cases consists of two lines each containing a name. The names will contain no more than **30** capital letters.

Output

For each of the test cases, you need to print one line of output. The output for each test case starts with the test case number, followed by the shortest length of the string and the number of unique strings that satisfies the given conditions.

You can assume that the number of unique strings will always be less than 2^{63} . Look at the sample output for the exact format.

Sample Input	Output for Sample Input
3 USA USSR LAILI MAJNU SHAHJAHAN MOMTAJ	Case 1: 5 3 Case 2: 9 40 Case 3: 13 15

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
const long long inf = 1LL << 40;
const int MN = 33;
pair<long long, long long> dp[MN][MN];
pair<long long, long long> go(const string &a, const string &b, int i, int j)
{
    if (dp[i][j].first != -1)
        return dp[i][j];
    if (i >= a.size() && j < b.size())
        return dp[i][j] = make_pair(b.size() - j, 1);
    if (i < a.size() && j >= b.size())
        return dp[i][j] = make_pair(a.size() - i, 1);
    if (i >= a.size() && j >= b.size())
        return dp[i][j] = make_pair(0, 1);
    long long best = inf;
    if (a[i] == b[j])
        best = min(best, go(a, b, i + 1, j + 1).first);
    best = min(best, go(a, b, i + 1, j).first);
    best = min(best, go(a, b, i, j + 1).first);
    long long ways = 0;
    if (a[i] == b[j]) {
        if (go(a, b, i + 1, j + 1).first == best)
            ways += go(a, b, i + 1, j + 1).second;
    }
    else {
        if (go(a, b, i + 1, j).first == best)
            ways += go(a, b, i + 1, j).second;
        if (go(a, b, i, j + 1).first == best)
            ways += go(a, b, i, j + 1).second;
    }
    return dp[i][j] = make_pair(best + 1, ways);
}

void solve() {
    string a, b;
    cin >> a >> b;
    cout << go(a, b, 0, 0).first << " " << go(a, b, 0, 0).second << endl;
}

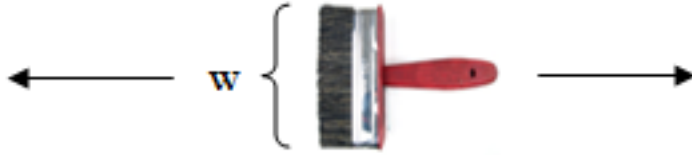
int main() {
    int n;
    cin >> n;
    for (int tc = 0; tc < n; ++tc) {
        cout << "Case " << tc + 1 << ": ";
    }
}
```

```
    for (int i = 0; i < MN; ++i) {  
        for (int j = 0; j < MN; ++j) {  
            dp[i][j] = make_pair(-1, -1);  
        }  
    }  
    solve();  
}  
return 0;  
}
```

1017 - Brush (III)

Time Limit: 2 second(s)

Memory Limit: 32 MB



Samir returned home from the contest and got angry after seeing his room dusty. Who likes to see a dusty room after a brain

storming programming contest? After checking a bit he found a brush in his room which has width w . Dusts are defined as 2D points. And since they are scattered everywhere, Samir is a bit confused what to do. He asked Samee and found his idea. So, he attached a rope with the brush such that it can be moved horizontally (in X axis) with the help of the rope but in straight line. He places it anywhere and moves it. For example, the y co-ordinate of the bottom part of the brush is 2 and its width is 3, so the y coordinate of the upper side of the brush will be 5. And if the brush is moved, all dusts whose y co-ordinates are between 2 and 5 (inclusive) will be cleaned. After cleaning all the dusts in that part, Samir places the brush in another place and uses the same procedure. He defined a **move** as placing the brush in a place and cleaning all the dusts in the horizontal zone of the brush.

You can assume that the rope is sufficiently large. Since Samir is too lazy, he doesn't want to clean all the room. Instead of doing it he thought that he would use at most k moves. Now he wants to find the maximum number of dust units he can clean using at most k moves. Please help him.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a blank line. The next line contains three integers N ($1 \leq N \leq 100$), w ($1 \leq w \leq 10000$) and k ($1 \leq k \leq 100$). N means that there are N dust points. Each of the next N lines contains two integers: x_i y_i denoting the coordinates of the dusts. You can assume that ($-10^9 \leq x_i, y_i \leq 10^9$) and all points are distinct.

Output

For each case print the case number and the maximum number of dusts Samir can clean using at most k moves.

Sample Input	Output for Sample Input
2 3 2 1 0 0 20 2 30 2	Case 1: 3 Case 2: 2

3	1	1
0	0	
20	2	
30	2	

SOLUTION

```
#include <bits/stdc++.h>
using namespace std;
const int inf = 1 << 30;
const int MN = 111;
int nxt[MN], cnt[MN];
int dp[MN][MN];
int go(const vector<int> &x, int i, int k) {
    if (k < 0)
        return -inf;
    if (i >= x.size())
        return 0;
    if (dp[i][k] != -1)
        return dp[i][k];
    return dp[i][k] = max(go(x, nxt[i], k - 1) + cnt[i], go(x, i + 1,
k));
}
void solve() {
    int n, w, k;
    scanf("%d%d%d", &n, &w, &k);
    vector<int> x(n), y(n);
    for (int i = 0; i < n; ++i) {
        scanf("%d%d", &x[i], &y[i]);
    }
    sort(y.begin(), y.end());
    for (int i = 0; i < n; ++i) {
        int j;
        for (j = i; j < n && ((y[j] - y[i]) <= w); ++j);
        nxt[i] = j;
        cnt[i] = j - i;
    }
    memset(dp, -1, sizeof dp);
    printf("%d\n", go(y, 0, k));
}
int main() {
    int n;
    cin >> n;
    for (int tc = 0; tc < n; ++tc) {
        printf("Case %d: ", tc + 1);
        solve();
    }
    return 0;
}
```


1018 - Brush (IV)

Time Limit: 2 second(s)

Memory Limit: 32 MB

Mubashwir returned home from the contest and got angry after seeing his room dusty. Who likes to see a dusty room after a brain storming programming contest? After checking a bit he found an old toothbrush in his room. Since the dusts are scattered everywhere, he is a bit confused what to do. So, he called Shakib. Shakib said that, 'Use the brush recursively and clean all the dust, I am cleaning my dust in this way!'

So, Mubashwir got a bit confused, because it's just a tooth brush. So, he will move the brush in a straight line and remove all the dust. Assume that the tooth brush only removes the dusts which lie on the line. But since he has a tooth brush so, he can move the brush in any direction. So, he counts a move as driving the tooth brush in a straight line and removing the dusts in the line.

Now he wants to find the maximum number of moves to remove all dusts. You can assume that dusts are defined as **2D** points, and if the brush touches a point, it's cleaned. Since he already had a contest, his head is messy. That's why he wants your help.

Input

Input starts with an integer **T** (≤ 1000), denoting the number of test cases.

Each case starts with a blank line. The next line contains three integers **N** ($1 \leq N \leq 16$). **N** means that there are **N** dust points. Each of the next **N** lines will contain two integers x_i y_i denoting the coordinate of a dust unit. You can assume that ($-1000 \leq x_i, y_i \leq 1000$) and all points are distinct.

Output

For each case print the case number and the minimum number of moves.

Sample Input	Output for Sample Input
2 3 0 0 1 1 2 2 3 0 0 1 1 2 3	Case 1: 1 Case 2: 2

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#define set(x, i) (x | (1 << i))
#define check(x, i) (x & (1 << i))
using namespace std;
enum CONST {N = 16};
typedef struct node {
    int x;
    int y;
} node;
int n;
int collinear[N+3][N+3];
int dp[1 << N+2];
node a[N + 5];
int cal(int state)
{
    int count;
    count = 0;
    for (int i = 1; i <= n; i++)
        if(check(state, i))
            count++;
    return count;
}
int explore(int status)
{
    int count;
    int temp;
    int mini;
    mini = INT_MAX;
    if(dp[status] != -1) {
        return dp[status];
    }
    count = cal(status);
    if(count == n-1) {
        return 1;
    }
    if(count == n) {
        return 0;
    }
}
```

```

        for (int i = 1; i <= n; i++) {
            if(check(status,i) == 0) {
                for (int j = i+1; j <= n; j++) {
                    if(check(status, j) == 0) {
                        temp = set(status, i);
                        temp = set(temp, j);
                        temp = temp | collinear[i][j];
                        mini = min(mini, explore(temp));
                    }
                }
                break;
            }
        }
        return dp[status] = 1 + mini;
    }
    bool is_collinear(int i, int j, int k)
    {
        int x1;
        int y1;
        int x2;
        int y2;
        int x3;
        int y3;
        x1 = a[i].x;
        y1 = a[i].y;
        x2 = a[j].x;
        y2 = a[j].y;
        x3 = a[k].x;
        y3 = a[k].y;
        return (y2 - y1) * (x3 - x2) == (x2 - x1) * (y3 - y2);
    }
    void calculate_collinear_points()
    {
        for (int i = 1; i <= n; i++) {
            for (int j = i + 1; j <= n; j++) {
                for (int k = 1; k <= n; k++) {
                    if(i != k and j != k and is_collinear(i, j, k))
                    {
                        collinear[i][j] = set(collinear[i][j],
k);
                        collinear[j][i] = set(collinear[j][i],
k);
                    }
                }
            }
        }
    }
}

```

```

}
int main()
{
    int t;
    node temp;
    scanf("%d", &t);
    for(int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);

        memset(collinear, 0, sizeof collinear);
        memset(dp, -1, sizeof dp);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &temp.x);
            scanf("%d", &temp.y);
            a[i] = temp;
        }
        calculate_collinear_points();
        printf("Case %d: %d\n", cs, explore(0));
    }
}

```

1021 - Painful Bases

Time Limit: 2 second(s)

Memory Limit: 32 MB

As you know that sometimes base conversion is a painful task. But still there are interesting facts in bases.

For convenience let's assume that we are dealing with the bases from 2 to 16. The valid symbols are **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F**. And you can assume that all the numbers given in this problem are valid. For example **67AB** is not a valid number of base **11**, since the allowed digits for base **11** are **0** to **A**.

Now in this problem you are given a base, an integer **K** and a valid number in the base which contains distinct digits. You have to find the number of permutations of the given number which are divisible by **K**. **K** is given in decimal.

For this problem, you can assume that numbers with leading zeroes are allowed. So, **096** is a valid integer.

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case starts with a blank line. After that there will be two integers, **base** ($2 \leq \text{base} \leq 16$) and **K** ($1 \leq K \leq 20$). The next line contains a valid integer in that base which contains distinct digits, that means in that number no digit occurs more than once.

Output

For each case, print the case number and the desired result.

Sample Input	Output for Sample Input
3	Case 1: 1
2 2	Case 2: 12
10	Case 3: 20922789888000
10 2	
5681	
16 1	
ABCDEF0123456789	

SOLUTION

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <stdio.h>
#include <string.h>
#define unset(x, i) (x & ~(1 << i))
#define set(x, i) (x | ((1 << i)))
#define check(x, i) (x & (1 << i))
using namespace std;
int base;
int by;
int n;
int a[30];
int rem[25];
int l;
unsigned long long dp[1 << 16][20];
unsigned long long pow[20];
bool vis[1 << 16];
unsigned long long explore(unsigned long long mask)
{
    if(vis[mask] == 1 or mask == 0) {
        return 0;
    }

    vis[mask] = 1;
    for (int i = 0; i < by; i++)
        dp[mask][i] = 0;
    for (int i = 0; i < base; i++) {
        if(check(mask, i)) {
            l--;
            explore(unset(mask, i));
            l++;
            for (int j = 0; j < by; j++) {
                dp[mask][((pow[l - 1] * i + j) % by) +=
dp[unset(mask, i)][j];
            }
        }
    }
}

int main()
{
```

```

int t;
char c;
char temp[20];
unsigned long long mask;
scanf("%d", &t);

for (int cs = 1; cs <= t; cs++) {
    scanf("%d", &base);
    scanf("%d", &by);
    scanf("%s", temp);

    n = strlen(temp);
    l = n;
    pow[0] = 1;
    mask = 0;
    memset(dp, 0, sizeof dp);
    memset(vis, 0, sizeof vis);
    for (int i = 1; i <= n; i++) {
        pow[i] = pow[i-1] * base;
        pow[i] = pow[i] % by;
    }

    for (int i = 0; i < n; i++) {
        if(isalpha(temp[i])) {
            a[i] = (temp[i] - 'A') + 10;
        }
        else {
            a[i] = (temp[i] - '0');
        }
        mask = set(mask, a[i]);
    }
    dp[0][0] = 1;
    explore(mask);
    printf("Case %d: %llu\n", cs, dp[mask][0]);
}
}

```

1025 - The Specials Menu

Time Limit: 2 second(s)

Memory Limit: 32 MB

Feuzem is an unemployed computer scientist who spends his days working at odd-jobs. While on the job he always manages to find algorithmic problems within mundane aspects of everyday life.

Today, while writing down the specials menu at the restaurant he's working at, he felt irritated by the lack of palindromes (strings which stay the same when reversed) on the menu. Feuzem is a big fan of palindromic problems, and started thinking about the number of ways he could remove letters from a particular word so that it would become a palindrome.

Two ways that differ due to order of removing letters are considered the same. And it can also be the case that no letters have to be removed to form a palindrome.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case contains a single word **W** ($1 \leq \text{length}(W) \leq 60$).

Output

For each case, print the case number and the total number of ways to remove letters from **W** such that it becomes a palindrome.

Sample Input	Output for Sample Input
3 SALADS PASTA YUMMY	Case 1: 15 Case 2: 8 Case 3: 11

SOLUTION

```
#include <iostream>
#include <string.h>
#include <stdio.h>
using namespace std;
char str[100];
long long dp[100][100];
long long wow(int i, int j)
{
    if(j < i) {
        return 0;
    }
    if(i == j) {
        return 1;
    }
    if(dp[i][j] != -1) {
        return dp[i][j];
    }
    if(str[i] == str[j]) {
        return dp[i][j] = 1 + wow(i+1, j) + wow(i, j-1);
    }
    else {
        return dp[i][j] = wow(i+1, j) + wow(i, j-1) - wow(i+1, j-1);
    }
}

int main()
{
    int t;
    int n;
    scanf("%d", &t);
    for (int i = 1; i <= t; i++) {
        scanf("%s", str);
        memset(dp, -1, sizeof(dp));
        n = strlen(str) - 1;
        printf("Case %d: %lld\n", i, wow(0, n));
    }
}
```

1027 - A Dangerous Maze

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are in a maze; seeing n doors in front of you in beginning. You can choose any door you like. The probability for choosing a door is equal for all doors.

If you choose the i^{th} door, it can either take you back to the same position where you begun in x_i minutes, or can take you out of the maze after x_i minutes. If you come back to the same position, you can't remember anything. So, every time you come to the beginning position, you have no past experience.

Now you want to find the expected time to get out of the maze.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case contains a blank line and an integer n ($1 \leq n \leq 100$) denoting the number of doors. The next line contains n space separated integers. If the i^{th} integer (x_i) is positive, you can assume that the i^{th} door will take you out of maze after x_i minutes. If it's negative, then the i^{th} door will take you back to the beginning position after $\text{abs}(x_i)$ minutes. You can safely assume that $1 \leq \text{abs}(x_i) \leq 10000$.

Output

For each case, print the case number and the expected time to get out of the maze. If it's impossible to get out of the maze, print '**inf**'. Print the result in p/q format. Where p is the numerator of the result and q is the denominator of the result and they are relatively prime. See the samples for details.

Sample Input	Output for Sample Input
3	Case 1: 1/1
1	Case 2: inf
1	Case 3: 18/1
2	
-10 -3	
3	
3 -6 -9	

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <utility>
#include <algorithm>
using namespace std;
pair <int, int> minify(pair <int, int> temp)
{
    int x;
    int y;
    int ok;
    x = temp.first;
    y = temp.second;
    ok = 1;
    while(ok) {
        ok = 0;
        for (int i = 2; i <= y; i++) {
            if(x % i == 0 and y % i == 0) {
                ok = 1;
                x = x / i;
                y = y / i;
            }
        }
    }
    temp.first = x;
    temp.second = y;
    return temp;
}

int main()
{
    int t;
    int psum;
    int nsum;
    int n;
    int ncount;
    int x;
    int y;

    pair <int, int> temp;
    scanf("%d", &t);
    for (int cases = 1; cases <= t; cases++) {
        scanf("%d", &n);
        nsum = 0;
        psum = 0;
```

```

ncount = 0;
for(int i = 0; i < n; i++) {
    scanf("%d", &x);
    if(x < 0) {
        ncount++;
        nsum += abs(x);
    }
    else {
        nsum += x;
    }
}
if(ncount == n) {
    printf("Case %d: inf\n", cases);
}
else {
    temp.first = nsum;
    temp.second = n - ncount;
    temp = minify(temp);
    printf("Case %d: %d/%d\n", cases, temp.first,
temp.second);
}
}
}

```

1030 - Discovering Gold

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are in a cave, a long cave! The cave can be represented by a $1 \times N$ grid. Each cell of the cave can contain any amount of gold.

Initially you are in position 1. Now each turn you throw a perfect 6 sided dice. If you get X in the dice after throwing, you add X to your position and collect all the gold from the new position. If your new position is outside the cave, then you keep throwing again until you get a suitable result. When you reach the N^{th} position you stop your journey. Now you are given the information about the cave, you have to find out the **expected** number of gold you can collect using the given procedure.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case contains a blank line and an integer N ($1 \leq N \leq 100$) denoting the dimension of the cave. The next line contains N space separated integers. The i^{th} integer of this line denotes the amount of gold you will get if you come to the i^{th} cell. You may safely assume that all the given integers will be non-negative and no integer will be greater than 1000.

Output

For each case, print the case number and the expected number of gold you will collect. Errors less than 10^{-6} will be ignored.

Sample Input	Output for Sample Input
3 1 101 2 10 3 3 3 6 9	Case 1: 101.0000000000 Case 2: 13.000 Case 3: 15

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
int main()
{
    double pro[110];
    int val[110];
    int t;
    int n;
    int j;
    int count;
    double exp;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        memset(pro, 0, sizeof pro);
        pro[1] = 1;
        for (int i = 1; i <= n; i++) {
            scanf("%d", val + i);
        }
        exp = val[1];
        for (int i = 2; i <= n; i++) {
            for (int k = 1; k <= 6 and i - k >= 1; k++) {
                j = i - k;
                count = min(6, n - j);
                pro[i] = pro[i] + (pro[j] * ( double(1) /
double(count))));
            }
            exp = exp + pro[i] * val[i];
        }
        printf("Case %d: %lf\n", cs, exp);
    }
}
```

1031 - Easy Game

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are playing a two player game. Initially there are n integer numbers in an array and player **A** and **B** get chance to take them alternatively. Each player can take one or more numbers from the left or right end of the array but cannot take from both ends at a time. He can take as many consecutive numbers as he wants during his time. The game ends when all numbers are taken from the array by the players. The point of each player is calculated by the summation of the numbers, which he has taken. Each player tries to achieve more points from other. If both players play optimally and player **A** starts the game then how much more point can player **A** get than player **B**?

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case contains a blank line and an integer **N** ($1 \leq N \leq 100$) denoting the size of the array. The next line contains **N** space separated integers. You may assume that no number will contain more than 4 digits.

Output

For each test case, print the case number and the maximum difference that the first player obtained after playing this game optimally.

Sample Input	Output for Sample Input
2 4 4 -10 -20 7 4 1 2 3 4	Case 1: 7 Case 2: 10

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <limits.h>
using namespace std;
long long dp[110][110][2];
int a[110];
int n;
long long explore(int l, int r, int chance)
{
    long long x;
    long long sum;
    if(l > r) {
        return 0;
    }
    if(dp[l][r][chance] != -1) {
        return dp[l][r][chance];
    }
    if(chance == 0) {
        sum = 0;
        x = INT_MIN;

        for (int i = l; i <= r; i++) {
            sum = sum + a[i];
            x = max(x, sum + explore(i+1, r, 1));
        }
        sum = 0;
        for (int i = r; i >= l; i--) {
            sum = sum + a[i];
            x = max(x, sum + explore(l, i-1, 1));
        }
        dp[l][r][chance] = x;
        return x;
    }
    if(chance == 1) {
        sum = 0;
        x = INT_MAX;

        for (int i = l; i <= r; i++) {
            sum = sum + -a[i];
            x = min(x, sum + explore(i+1, r, 0));
        }
    }
}
```

```

        sum = 0;
        for (int i = r; i >= l; i--) {
            sum = sum + -a[i];
            x = min(x, sum + explore(l, i-1, 0));
        }
        dp[l][r][chance] = x;
        return x;
    }
}

int main()
{
    int t;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {

        memset(dp, -1, sizeof dp);
        scanf("%d", &n);

        for (int i = 0; i < n; i++) {
            scanf("%d", a + i);
        }
        printf("Case %d: %lld\n", cs, explore(0, n-1, 0));
    }
}

```

1032 - Fast Bit Calculations

Time Limit: 2 second(s)

Memory Limit: 32 MB

A bit is a binary digit, taking a logical value of either **1** or **0** (also referred to as "true" or "false" respectively). And every decimal number has a binary representation which is actually a series of bits. If a bit of a number is **1** and its next bit is also **1** then we can say that the number has a **1** adjacent bit. And you have to find out how many times this scenario occurs for all numbers up to **N**.

Examples:

Number	Binary	Adjacent Bits
12	1100	1
15	1111	3
27	11011	2

Input

Input starts with an integer **T** (≤ 10000), denoting the number of test cases.

Each case contains an integer **N** ($0 \leq N < 2^{31}$).

Output

For each test case, print the case number and the summation of all adjacent bits from **0** to **N**.

Sample Input	Output for Sample Input
7	Case 1: 0
0	Case 2: 2
6	Case 3: 12
15	Case 4: 13
20	Case 5: 13
21	Case 6: 14
22	Case 7: 16106127360
2147483647	

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
long long dp[33][3][33][3];
int n;
long long go(const int &mask, int i, bool prev, int cur, bool top) {
    if (i < 0)
        return cur;
    if (dp[i][prev][cur][top] != -1)
        return dp[i][prev][cur][top];
    long long ans = go(mask, i - 1, 0, cur, top && !((mask >> i) & 1));
    if (!top || (top && ((mask >> i) & 1)))
        ans += go(mask, i - 1, 1, cur + (prev == 1), top && ((mask >> i) & 1));
    return dp[i][prev][cur][top] = ans;
}
void solve() {
    int k;
    cin >> k;
    n = 0;
    for (int i = 0; i < 32; ++i)
        if ((k >> i) & 1)
            n = i;
    memset(dp, -1, sizeof dp);
    printf("%lld\n", go(k, n, 0, 0, 1));
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int tc;
    cin >> tc;
    for (int i = 0; i < tc; ++i) {
        printf("Case %d: ", i + 1);
        solve();
    }
    return 0;
}
```

1033 - Generating Palindromes

Time Limit: 2 second(s)

Memory Limit: 32 MB

By definition palindrome is a string which is not changed when reversed. **"MADAM"** is a nice example of palindrome. It is an easy job to test whether a given string is a palindrome or not. But it may not be so easy to generate a palindrome.

Here we will make a palindrome generator which will take an input string and return a palindrome. You can easily verify that for a string of length **n**, no more than **(n - 1)** characters are required to make it a palindrome. Consider **"abcd"** and its palindrome **"abcdcba"** or **"abc"** and its palindrome **"abcba"**. But life is not so easy for programmers!! We always want optimal cost. And you have to find the minimum number of characters required to make a given string to a palindrome if you are only allowed to insert characters at any position of the string.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case contains a string of lowercase letters denoting the string for which we want to generate a palindrome. You may safely assume that the length of the string will be positive and no more than **100**.

Output

For each case, print the case number and the minimum number of characters required to make string to a palindrome.

Sample Input	Output for Sample Input
6	Case 1: 3
abcd	Case 2: 0
aaaa	Case 3: 2
abc	Case 4: 1
aab	Case 5: 0
abababaabababa	Case 6: 9
pqrsabcdpqr	

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;
int pali(string x, string y)
{
    int a[110][110];
    memset(a, 0, sizeof(a));

    for (int i = x.length(); i >= 0; i--) {
        for (int j = y.length(); j >= 0; j--) {
            if(i == x.length() or j == y.length()) {
                a[i][j] = 0;
                continue;
            }
            if(x[i] == y[j]) {
                a[i][j] = 1 + a[i+1][j+1];
                continue;
            }
            a[i][j] = max(a[i+1][j], a[i][j+1]);
        }
    }
    return x.length() - a[0][0];
}

int main()
{
    string x;
    string y;
    char temp[110];
    int t;
    scanf("%d", &t);

    for (int cases = 1; cases <= t; cases++) {
        scanf("%s", temp);
        x = temp;
        for (int i = 0, j = x.length() - 1; i < j; i++, j--) {
            swap(temp[i], temp[j]);
        }
        y = temp;
        printf("Case %d: %d\n", cases, pali(x, y));
    }
}
```


1036 - A Refining Company

Time Limit: 3 second(s)

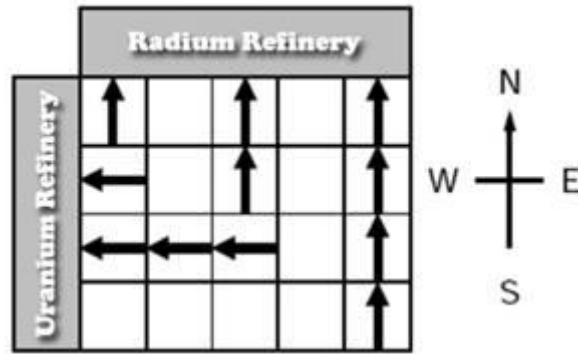
Memory Limit: 32 MB

Its year 2200, planet Earth is out of resources and people are relying on the resources from other planets. There are several Refining Companies who collect these resources from other planets and bring them back to Earth. The task may sound simple, but in reality it's a challenging job. The resources are scattered and after collecting them, they have to be taken to a place where they can be refined. Since some minerals are extremely dangerous, the whole process should be done very carefully. A single tiny mistake can cause a massive explosion resulting in a huge loss.

You work in such a company who collects Uranium and Radium from planet Krypton. These minerals are used for generating powers. For simplicity you have divided planet Krypton into cells that form a matrix of m rows and n columns, where the rows go from east to west and the columns go from north to south. Your advanced mine detector has detected the approximate amount of Radium and Uranium in each cell. Your company has built two refining factories, one in West and the other in North. The factory in North is used to refine Radium and the factory in West is used to refine Uranium. Your task is to design the conveyor belt system that will allow them to mine the largest amount of minerals.

There are two types of conveyor belts: the first moves minerals from east to west, the second moves minerals from south to north. In each cell you can build either type of conveyor belt, but you cannot build both of them in the same cell. If two conveyor belts of the same type are next to each other, then they can be connected. For example, the Radium mined at a cell can be transported to the Radium refinement factory via a series of south-north conveyor belts.

The minerals are very unstable, thus they have to be brought to the factories on a straight path without any turns. This means that if there is a south-north conveyor belt in a cell, but the cell north of it contains an east-west conveyor belt, then any mineral transported on the south-north conveyor belt will be lost. The minerals mined in a particular cell have to be put on a conveyor belt immediately; in the same cell (thus they cannot start the transportation in an adjacent cell). Furthermore, any Radium transported to the Uranium refinement factory will be lost, and vice versa.



Your program has to design a conveyor belt system that maximizes the total amount of minerals mined, i.e., the sum of the amount of Radium transported to the Radium refinery and the amount of Uranium to the Uranium refinery.

Input

Input starts with an integer **T** (≤ 10), denoting the number of test cases.

Each case begins with a blank line and two integers: **m** - the number of rows, and **n** - the number columns ($1 \leq m, n \leq 500$). The next **m** lines describe the amount of Uranium that can be found in the cells. Each of these **m** lines contains **n** integers. The first line corresponds to the northernmost row; the first integer of each line corresponds to the westernmost cell of the row. The integers are between **0** and **1000**. The next **m** lines describe in a similar fashion the amount of Radium found in the cells. Data set is huge, so use faster i/o methods.

Output

For each case of input you have to print the case number and the maximum amount of minerals you can collect.

Sample Input	Output for Sample Input
2 4 4 0 0 10 9 1 3 10 0 4 2 1 3 1 1 20 0 10 0 0 0 1 1 1 30 0 0 5 5 5 10 10 10 2 3 5 10 34 0 0 0 0 0 0 50 0 0	Case 1: 98 Case 2: 50

Note

Dataset is huge. Use faster I/O methods.

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
const int MN = 555;
int dp[MN][MN];
int go(vector<vector<int> > &a, vector<vector<int> > &b, int i, int j) {
    if (i < 0 || j < 0)
        return 0;
    if (dp[i][j] != -1)
        return dp[i][j];
    return dp[i][j] = max(go(a, b, i - 1, j) + a[i][j], go(a, b, i, j - 1) +
b[i][j]);
}
void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<int> > a(n, vector<int> (m)), b(n, vector<int> (m));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j) {
            cin >> a[i][j];
            if (j)
                a[i][j] += a[i][j - 1];
        }
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j) {
            cin >> b[i][j];
            if (i)
                b[i][j] += b[i - 1][j];
        }
    memset(dp, -1, sizeof dp);
    printf("%d\n", go(a, b, n - 1, m - 1));
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int tc;
    cin >> tc;
    for (int i = 0; i < tc; ++i) {
        printf("Case %d: ", i + 1);
        solve();
    }
    return 0;
}
```


1037 - Agent 47

Time Limit: 2 second(s)

Memory Limit: 32 MB

Agent **47** is in a dangerous Mission "Black Monster Defeat - 15". It is a secret mission and so 47 has a limited supply of weapons. As a matter of fact he has only one weapon the old weak "**KM .45 Tactical (USP)**". The mission sounds simple - "he will encounter at most 15 Targets and he has to kill them all". The main difficulty is the weapon. After huge calculations, he found a way out. That is after defeating a target, he can use target's weapon to kill other targets. So there must be an order of killing the targets so that the total number of weapon shots is minimized. As a personal programmer of Agent 47 you have to calculate the least number of shots that need to be fired to kill all the targets.



Agent 47

Now you are given a list indicating how much damage each weapon does to each target per shot, and you know how much health each target has. When a target's health is reduced to **0** or less, he is killed. **47** start off only with the **KM .45 Tactical (USP)**, which does damage **1** per shot to any target. The list is represented as a **2D** matrix with the i^{th} element containing **N** single digit numbers ('0'-'9'), denoting the damage done to targets **0, 1, 2, ..., N-1** by the weapon obtained from target **i**, and the health is represented as a series of **N** integers, with the i^{th} element representing the amount of health that target has.

Given the list representing all the weapon damages, and the health each target has, you should find the least number of shots he needs to fire to kill all of the targets.

Input

Input starts with an integer **T** (≤ 40), denoting the number of test cases.

Each case begins with a blank line and an integer **N** ($1 \leq N \leq 15$). The next line contains **N** space separated integers between **1** and 10^6 denoting the health of the targets **0, 1, 2, ..., N-1**. Each of the next **N** lines contains **N** digits. The j^{th} digit of the i^{th} line denotes the damage done to target **j**, if you use the weapon of target **i** in each shot.

Output

For each case of input you have to print the case number and the least number of shots that need to be fired to kill all of the targets.

Sample Input	Output for Sample Input
2 3 10 10 10 010 100 111 3 3 5 7 030 500 007	Case 1: 30 Case 2: 12

SOLUTION

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <iostream>
#include <limits.h>
#define check(n, pos) (n & (1<<pos))
#define set(n, pos) (n | (1<<pos))
using namespace std;
int power[20];
int n;
int x[20][20];
int dp[1 << 17];
int unset(int a, int i)
{
    int mask;
    mask = 1 << i;
    a = a & ~mask;
    return a;
}
int wow(int count, int a)
{
    if(count == n) {
        return 0;
    }
    if(dp[a] != -1) {
        return dp[a];
    }
    int maxi;
    int c;
    maxi = INT_MAX;
    for (int i = 0; i < n; i++) {
        if(check(a, i) == 0) {
            for (int j = 0; j <= n; j++) {
                if(check(a, j) != 0 and x[j][i] != 0) {
                    a = set(a, i);
                    c = (int) ceil((float) power[i] /
(float) x[j][i]);
                    maxi = min(maxi, c + wow(count + 1, a));
                    a = unset(a, i);
                }
            }
        }
    }
    dp[a] = maxi;
    return dp[a];
}
```

```

        dp[a] = maxi;
        return maxi;
    }

int main()
{
    int t;
    int a;
    char c;
    scanf("%d", &t);
    for (int cases = 1; cases <= t; cases++) {
        scanf("%d", &n);
        memset(dp, -1, sizeof(dp));
        a = 0;
        a = set(a, n);
        for (int i = 0; i < n; i++) {
            scanf("%d", &power[i]);
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cin >> c;
                x[i][j] = (int) (c - '0');
            }
        }
        for (int i = 0; i < n; i++) {
            x[n][i] = 1;
        }
        printf("Case %d: %d\n", cases, wow(0, a));
    }
}

```

1038 - Race to 1 Again

Time Limit: 2 second(s)

Memory Limit: 32 MB

Rimi learned a new thing about integers, which is - any positive integer greater than 1 can be divided by its divisors. So, he is now playing with this property. He selects a number N . And he calls this D .

In each turn he randomly chooses a divisor of D (1 to D). Then he divides D by the number to obtain new D . He repeats this procedure until D becomes 1. What is the expected number of moves required for N to become 1.

Input

Input starts with an integer T (≤ 10000), denoting the number of test cases.

Each case begins with an integer N ($1 \leq N \leq 10^5$).

Output

For each case of input you have to print the case number and the expected value. Errors less than 10^{-6} will be ignored.

Sample Input	Output for Sample Input
3	Case 1: 0
1	Case 2: 2.00
2	Case 3: 3.0333333333
50	

SOLUTION

```
#include <iostream>
#include <stdio.h>
#define N 100000
using namespace std;
double pro[N + 4];
int count[N+ 4];
int main()
{
    double sum;
    double temp;
    int x;
    int n;
    for (int i = 2; i <= N; i++) {
        sum = pro[i];
        n = count[i];
        n += 2;
        sum += n;
        sum = sum / (double) (n - 1);
        pro[i] = sum;
        for (int j = 2 * i; j <= N; j = j + i) {
            count[j]++;
            pro[j] += sum;
        }
    }
    int t;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &x);
        printf("Case %d: %lf\n", cs, pro[x]);
    }
}
```

1044 - Palindrome Partitioning

Time Limit: 1 second(s)

Memory Limit: 32 MB

A palindrome partition is the partitioning of a string such that each separate substring is a palindrome.

For example, the string "ABACABA" could be partitioned in several different ways, such as {"A","B","A","C","A","B","A"}, {"A","BACAB","A"}, {"ABA","C","ABA"}, or {"ABACABA"}, among others.

You are given a string **s**. Return the minimum possible number of substrings in a palindrome partition of **s**.

Input

Input starts with an integer **T** (≤ 40), denoting the number of test cases.

Each case begins with a non-empty string **s** of uppercase letters with length no more than **1000**.

Output

For each case of input you have to print the case number and the desired result.

Sample Input	Output for Sample Input
3	Case 1: 1
AAAA	Case 2: 8
ABCDEFGH	Case 3: 5
QWERTYTREWQWERT	

SOLUTION

```
#include <stdio.h>
#include <iostream>
#include <string.h>
using namespace std;
int palin[1005][1005];
int generate(string x)
{
    int n;
    int e;
    n = x.length();
    for (int i = 0; i < n; i++) {
        palin[i][i] = 1;
    }
    for (int length = 2; length <= n; length++) {
        for (int s = 0; s <= n - length; s++) {
            e = s + length - 1;
            if(length == 2) {
                palin[s][e] = (x[s] == x[e]);
            }
            else {
                palin[s][e] = ((x[s] == x[e]) and palin[s+1][e-
1]);
            }
        }
    }
}

int main()
{
    int t;
    int explore[1001][1001];
    int n;
    scanf("%d", &t);
    for (int cases = 1; cases <= t; cases++) {

        int l;
        char inp[1005];
        string x;

        scanf("%s", inp);

        memset(explore, 0, sizeof(explore));
        memset(palin, 0, sizeof(palin));
        x = inp;
```

```

n = x.length();
generate(x);
for (int i = n; i >= 0; i--) {
    for (int j = n; j >= i; j--) {

        if(j == n) {
            explore[i][j] = j - i;
            continue;
        }

        if(palin[i][j]) {
            explore[i][j] = min( 1 + explore[j+1]
[j+1], explore[i][ j+1]);

            continue;
        }
        else {
            explore[i][j] = explore[i][j+1];
            continue;
        }
    }
}

printf("Case %d: %d\n", cases, explore[0][0]);
}
}

```

1047 - Neighbor House

Time Limit: 0.5 second(s)

Memory Limit: 32 MB

The people of Mohammadpur have decided to paint each of their houses red, green, or blue. They've also decided that no two neighboring houses will be painted the same color. The neighbors of house i are houses $i-1$ and $i+1$. The first and last houses are not neighbors.

You will be given the information of houses. Each house will contain three integers "**R G B**" (quotes for clarity only), where **R**, **G** and **B** are the costs of painting the corresponding house red, green, and blue, respectively. Return the minimal total cost required to perform the work.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case begins with a blank line and an integer n ($1 \leq n \leq 20$) denoting the number of houses. Each of the next n lines will contain 3 integers "**R G B**". These integers will lie in the range $[1, 1000]$.

Output

For each case of input you have to print the case number and the minimal cost.

Sample Input	Output for Sample Input
2 4 13 23 12 77 36 64 44 89 76 31 78 45 3 26 40 83 49 60 57 13 89 99	Case 1: 137 Case 2: 96

SOLUTION

```
#include <stdio.h>
#include <limits.h>
#include <iostream>
using namespace std;
int main()
{
    int n;
    int a[23][5];
    int ans[23][5];
    int x;
    int t;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < 3; j++) {
                scanf("%d", &a[i][j]);
            }
        }

        for(int i = n; i >= 0; i--) {
            for (int prev = 2; prev >= 0; prev--) {

                x = INT_MAX;
                if( i == n) {
                    ans[i][prev] = 0;
                    continue;
                }
                for (int j = 0; j < 3; j++) {
                    if(j != prev) {
                        x = min(x, a[i][j] +
ans[i+1][j]);
                    }
                }
                ans[i][prev] = x;
            }
        }
        printf("Case %d: %d\n",cs,  min(ans[0][0], min(ans[0][1],
ans[0][2])));
    }
}
```


1050 - Marbles

Time Limit: 2 second(s)

Memory Limit: 32 MB

Your friend Jim has challenged you to a game. He has a bag containing red and blue marbles. There will be an odd number of marbles in the bag, and you go first. On your turn, you reach into the bag and remove a random marble from the bag; each marble may be selected with equal probability. After your turn is over, Jim will reach into the bag and remove a blue marble; if there is no blue marble for Jim to remove, then he wins. If the final marble removed from the bag is blue (by you or Jim), you will win. Otherwise, Jim wins.

Given the number of red and blue marbles in the bag, determine the probability that you win the game.

Input

Input starts with an integer **T** (≤ 10000), denoting the number of test cases.

Each case begins with two integers **R** and **B** denoting the number of red and blue marbles respectively. You can assume that $0 \leq R, B \leq 500$ and **R+B** is odd.

Output

For each case of input you have to print the case number and your winning probability. Errors less than 10^{-6} will be ignored.

Sample Input	Output for Sample Input
5	Case 1: 0.3333333333
1 2	Case 2: 0.13333333
2 3	Case 3: 0.2285714286
2 5	Case 4: 0
11 6	Case 5: 0.1218337218
4 11	

SOLUTION

```
#include <bits/stdc++.h>
#define ld long double
#define ll long long int
#define mod 1000000007
#define ll_inf 1000000000000000
#define int_inf 1000000000
#define pb push_back
#define endl '\n'
#define Endl '\n'
#define eps 1e-9
#define PI acos(-1.0)
#define ii pair<int,int>
#define se second
#define fi first
using namespace std;
bool vis[505][505];
double dp[502][505];
double cal(int red, int blue){
    if(red==0){
        if(blue%2==0)return dp[red][blue]=0.0;
        return dp[red][blue]= 1.0;
    }
    if(blue==0)return dp[red][blue]=0.0;
    if(blue<=0)return 0.0;
    double &ret=dp[red][blue];
    if(vis[red][blue])return dp[red][blue];
    vis[red][blue]=1;
    ret=1.0;
    ret=((1.0*red)/(red+blue)*cal(red-1, blue-1));
    if(blue>=2)ret+=((1.0*blue)/(red+blue)*cal(red, blue-2));
    return ret;
}
int main(){
    cout.precision(12);
    int t, ca=1;
    cin>>t;
    while(t--){
        int red,blue;
        scanf("%d%d",&red, &blue);
        printf("Case %d: %.8f\n", ca++, cal(red, blue));
    }
}
```


1051 - Good or Bad

Time Limit: 2 second(s)

Memory Limit: 32 MB

A string is called bad if it has 3 vowels in a row, or 5 consonants in a row, or both. A string is called good if it is not bad. You are given a string s , consisting of uppercase letters ('A'-'Z') and question marks ('?'). Return "BAD" if the string is definitely bad (that means you cannot substitute letters for question marks so that the string becomes good), "GOOD" if the string is definitely good, and "MIXED" if it can be either bad or good.

The letters 'A', 'E', 'I', 'O', 'U' are vowels, and all others are consonants.

Input

Input starts with an integer T (≤ 200), denoting the number of test cases.

Each case begins with a non-empty string with length no more than 50.

Output

For each case of input you have to print the case number and the result according to the description.

Sample Input	Output for Sample Input
5	Case 1 : BAD
FFFF?EE	Case 2 : GOOD
HELLOWORLD	Case 3 : BAD
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Case 4 : MIXED
HELLO?ORLD	Case 5 : BAD
AAA	

SOLUTION

```
#include <iostream>
#include <vector>
#include <string.h>
#include <stdio.h>
using namespace std;
int ab;
int ok;
bool ans[55][1000];
vector <int> x;
string a;
int check(int n)
{
    int account = 0;
    int bcount = 0;
    int qcount = 0;
    int i;
    for ( i = 0; i < a.length(); i++) {
        if(a[i] == 'A' or a[i] == 'E' or a[i] == 'I' or a[i] == 'O'
or a[i] == 'U') {
            account++;
            bcount = 0;
            a[i] = 'A';
        }
        else {
            account = 0;
            if(a[i] != '?') {
                bcount++;
                a[i] = 'B';
            }
            else {
                bcount = 0;
                qcount++;
            }
        }
        if(account == 3 or bcount == 5) {
            break;
        }
    }
    if(account == 3 or bcount == 5) {
        return 0;
    }
}
```

```

        }
        else {
            return 1;
        }
    }
int explore(int i)
{

    int t;
    int temp;
    int mul = 1;
    temp = 0;
    if(i >= x.size()) {
        ok++;
        return 0;
    }
    t = x[i];
    a[t] = 'A';

    for (int j = 0; (t - j) >= 0 and j <= 5; j++) {
        temp = temp + ((a[t-j] - 'A') * mul);
        mul *= 2;
    }
    if(check(t)) {
        if(ans[i][temp] == 0) {
            explore(i+1);
            ans[i][temp] = 1;
        }
    }
    else {
        ab++;
    }

    mul = 1;
    a[t] = 'B';

    temp = 0;
    for (int j = 0; (t - j) >= 0 and j <= 5; j++) {
        temp += (a[t-j] - 'A') * mul;
        mul *= 2;
    }

    if(check(t)) {
        if(ans[i][temp] == 0) {
            explore(i+1);
            ans[i][temp] = 1;
        }
    }
}

```



```

        }
    }
    else {
        ab++;
    }
    a[t] = '?';
}

int main()
{
    int t;
    int account;
    int bcount;
    int qcount;
    int i;
    cin >> t;
    for (int cs = 1; cs <= t; cs++) {
        cin >> a;
        vector<int> temp;
        swap(x, temp);
        account = 0;
        bcount = 0;
        qcount = 0;
        for ( i = 0; i < a.length(); i++) {

            if(a[i] == 'A' or a[i] == 'E' or a[i] == 'I' or a[i]
== 'O' or a[i] == 'U') {

                account++;
                bcount = 0;
                a[i] = 'A';
            }
            else {
                account = 0;
                if(a[i] != '?') {
                    bcount++;
                    a[i] = 'B';
                }
                else {
                    bcount = 0;
                    qcount++;
                }
            }
            if(account == 3 or bcount == 5) {
                break;
            }

```

```

    }
    if(qcount == 0) {
        if(account == 3 or bcount == 5) {
            printf("Case %d: BAD\n", cs);
        }
        else {
            printf("Case %d: GOOD\n", cs);
        }
    }
    else {
        for (int i = 0; i < a.length(); i++) {
            if(a[i] == '?') {
                x.push_back(i);
            }
        }
        ab = 0;
        ok = 0;
        memset(ans, 0, sizeof(ans));
        explore(0);

        if(ab and ok) {
            printf("Case %d: MIXED\n", cs);
        }
        if(ab and ok == 0) {
            printf("Case %d: BAD\n", cs);
        }
        if(ab == 0 and ok) {
            printf("Case %d: GOOD\n", cs);
        }
    }
}
}
}

```

1057 - Collecting Gold

Time Limit: 2 second(s)

Memory Limit: 32 MB

Finally you found the city of Gold. As you are fond of gold, you start collecting them. But there are so much gold that you are getting tired collecting them.

So, you want to find the minimum effort to collect all the gold.

You can describe the city as a **2D** grid, where your initial position is marked by an '**x**'. An empty place will be denoted by a '.'. And the cells which contain gold will be denoted by '**g**'. In each move you can go to all **8** adjacent places inside the city.

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case will start with a blank line and two integers, **m** and **n** ($0 < m, n < 20$) denoting the row and columns of the city respectively. Each of the next **m** lines will contain **n** characters describing the city. There will be exactly one '**x**' in the city and at most **15** gold positions.

Output

For each case of input you have to print the case number and the minimum steps you have to take to collect all the gold and go back to '**x**'.

Sample Input	Output for Sample Input
2 5 5 x..... g..... g..... g..... 5 5 x..... g..... g.....	Case 1: 8 Case 2: 4

SOLUTION

```
#include <iostream>
#include <vector>
#include <utility>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
using namespace std;
int n;
int k;
int vis[20];
int a[20][20];
int ans[20][(1 << 16) + 1];
int check(int x, int i)
{
    return (x & (1 << i));
}
int set(int x, int i)
{
    x = x ^ (1 << i);
    return x;
}
int explore(int i, int x)
{
    int k;
    int temp;
    k = 1 << (n + 1);
    k--;
    if(x == k) {
        return a[i][0];
    }

    if(ans[i][x] != -1) {
        return ans[i][x];
    }
    int mini = INT_MAX;
    for (int j = 1; j <= n; j++) {
        if(vis[j] == 0) {
            x = set(x, j);
            vis[j] = 1;
            temp = a[i][j] + explore(j, x);
            mini = min(mini, temp);
            x = set(x, j);
        }
    }
    ans[i][x] = mini;
    return mini;
}
```

```

        vis[j] = 0;
    }
}
ans[i][x] = mini;
return mini;
}
int main()
{
    int r;
    int c;
    int t;
    char cc;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &r);
        scanf("%d", &c);
        vector <pair <int, int> > gr(20);
        n = 0;
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                cin >> cc;

                if(cc == 'x') {
                    gr[0] = make_pair(i, j);
                    continue;
                }
                if(cc == 'g') {
                    n++;
                    gr[n] = make_pair(i, j);
                }
            }
        }
        for (int i = 0; i <= n; i++) {
            for (int j = i + 1; j <= n; j++) {
                a[i][j] = a[j][i] = max(abs(gr[i].first -
gr[j].first) , abs(gr[i].second - gr[j].second));
            }
        }

        memset(ans, -1, sizeof(ans));
        printf("Case %d: %d\n",cs, explore(0, 1));
    }
}

```

1060 - nth Permutation

Time Limit: 2 second(s)

Memory Limit: 32 MB

Given a string of characters, we can permute the individual characters to make new strings. At first we order the string into alphabetical order. Then we start permuting it.

For example the string 'abba' gives rise to the following 6 distinct permutations in alphabetical order.

aabb 1
abab 2
abba 3
baab 4
baba 5
bbaa 6

Given a string, you have to find the n^{th} permutation for that string. For the above case 'aabb' is the 1st and 'baab' is the 4th permutation.

Input

Input starts with an integer T (≤ 200), denoting the number of test cases.

Each case contains a non empty string of lowercase letters with length no more than 20 and an integer n ($0 < n < 2^{31}$).

Output

For each case, output the case number and the n^{th} permutation. If the n^{th} permutation doesn't exist print 'Impossible'.

Sample Input	Output for Sample Input
3 aabb 1 aabb 6 aabb 7	Case 1: aabb Case 2: bbaa Case 3: Impossible

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
long long fact[22];
long long ways(vector<int> &c) {
    long long den = 1, total = 0;
    for (int i = 0; i < c.size(); ++i) {
        den *= fact[c[i]];
        total += c[i];
    }
    return fact[total] / den;
}
void solve() {
    string line;
    int n;
    cin >> line >> n;
    vector<int> frec(30);
    for (int i = 0; i < line.size(); ++i)
        frec[line[i] - 'a']++;
    if (ways(frec) < n) {
        puts("Impossible");
        return;
    }
    for (int i = 0; i < line.size(); ++i) {
        for (int j = 0; j < frec.size(); ++j) {
            if (frec[j]) {
                frec[j]--;
                if (ways(frec) >= n) {
                    putchar('a' + j);
                    break;
                } else {
                    n -= ways(frec);
                    frec[j]++;
                }
            }
        }
    }
    putchar('\n');
}
int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    fact[0] = 1;
    for (long long i = 1; i < 22; ++i)
```

```
        fact[i] = fact[i - 1] * i;
    int tc;
    cin >> tc;
    for (int i = 0; i < tc; ++i) {
        printf("Case %d: ", i + 1);
        solve();
    }
    return 0;
}
```


1061 - N Queen Again

Time Limit: 2 second(s)

Memory Limit: 32 MB

Given an **8 x 8** chess board where **8** queens are placed in any arbitrary order. You want to move the queens such that no one attacks each other.

In each move you can move a queen as in normal chess. That means you can move a queen vertically, horizontally or diagonally. And you can move it to multiple cells at a time but the direction can't be changed and you can't jump over any other queen. Now you want to find the minimum number of moves to do this.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case contains a blank line and an **8 x 8** board consisting of '.' and exactly eight 'q'. '.' stands for empty position and 'q' stands for a queen.

Output

For each case, output the case number and minimum number of queen moves you have to do such that no queen attacks another.

Sample Input	Output for Sample Input
2 q..... .q..... ..q..... ...q....q...q..q.q qqq..q. .q...q..q.. ..q.....	Case 1: 7 Case 2: 5

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
const int MN = 666;
const int inf = 1 << 30;
struct pos{
    int x, y;
    pos() {}
    pos(int a, int b) : x(a), y(b) {}
};
pos solutions[MN][8];
int num_sol;
bool empty(int &mask, int &index) {
    return ((mask >> index) & 1) == 0;
}
void precompute(pos *cur, int row, int cols, int d1, int d2) {
    if (row == 8) {
        for (int i = 0; i < 8; ++i) {
            solutions[num_sol][i] = cur[i];
        }
        num_sol++;
        return;
    }
    for (int i = 0; i < 8; ++i) {
        int right = row + i;
        int left = 7 - row + i;
        if (empty(cols, i) && empty(d1, right) && empty(d2, left)) {
            cur[row] = pos(row, i);
            precompute(cur, row + 1, cols | (1 << i), d1 | (1 << right), d2 | (1 <<
left));
        }
    }
}
int cost(pos &a, pos &b) {
    int dx = a.x - b.x;
    int dy = a.y - b.y;
    if (dx == 0 && dy == 0) return 0;
    if (dx == 0 || dy == 0) return 1;
    if (fabs(dx) == fabs(dy)) return 1;
    return 2;
}
int dp[(1 << 8) + 10];
int go(pos *target, pos *cur, int mask) {
    if (dp[mask] != -1)
```

```

        return dp[mask];
    if (mask == (1 << 8) - 1)
        return dp[mask] = 0;
    int best = inf;
    int queen = __builtin_popcount(mask);
    for (int i = 0; i < 8; ++i) {
        if(!((mask >> i) & 1)) {
            best = min(best, go(target, cur, mask | (1 << i)) + cost(target[i],
cur[queen]));
        }
    }
    return dp[mask] = best;
}

void solve() {
    pos board[8];
    char t;
    int q = 0;
    for (int i = 0; i < 8; ++i) {
        for (int j = 0; j < 8; ++j) {
            cin >> t;
            if (t == 'q') {
                board[q++] = pos(i, j);
            }
        }
    }
    int ans = inf;
    for (int i = 0; i < num_sol; ++i) {
        memset(dp, -1, sizeof dp);
        ans = min(ans, go(solutions[i], board, 0));
    }
    printf("%d\n", ans);
}

int main() {
    ios_base::sync_with_stdio(false);cin.tie(NULL);
    num_sol = 0;
    pos cur[8];
    precompute(cur, 0, 0, 0, 0);
    int tc;
    cin >> tc;
    for (int i = 0; i < tc; ++i) {
        printf("Case %d: ", i + 1);
        solve();
    }
    return 0;
}

```


1064 - Throwing Dice

Time Limit: 2 second(s)

Memory Limit: 32 MB

n common cubic dice are thrown. What is the probability that the sum of all thrown dice is at least x ?

Input

Input starts with an integer T (≤ 200), denoting the number of test cases.

Each test case contains two integers n ($1 \leq n < 25$) and x ($0 \leq x < 150$). The meanings of n and x are given in the problem statement.

Output

For each case, output the case number and the probability in ' p/q ' form where p and q are relatively prime. If q equals 1 then print p only.

Sample Input	Output for Sample Input
7	Case 1: 20/27
3 9	Case 2: 0
1 7	Case 3: 1
24 24	Case 4: 11703055/78364164096
15 76	Case 5: 25/4738381338321616896
24 143	Case 6: 1/2
23 81	Case 7: 55/46656
7 38	

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
#define endl '\n'
unsigned long long p6 [] = { 1LL, 6LL, 36LL, 216LL, 1296LL, 7776LL, 46656LL,
                             279936LL, 1679616LL, 10077696LL,
                             60466176LL, 362797056LL, 2176782336LL, 13060694016LL,
                             78364164096LL,
                             470184984576LL, 2821109907456LL, 16926659444736LL,
                             101559956668416LL,
                             609359740010496LL, 3656158440062976LL,
                             21936950640377856LL,
                             131621703842267136LL, 789730223053602816LL,
                             4738381338321616896LL };
void solve() {
    int n, x;
    cin >> n >> x;
    vector<vector<unsigned long long> > dp(n + 1, vector<unsigned long long>
(x + 1, 0));
    dp[0][x] = 1;
    for (int i = 1; i <= n; ++i)
        for (int j = 0; j <= x; ++j)
            for (int k = 1; k <= 6; ++k) {
                int next = min(j + k, x);
                dp[i][j] += dp[i - 1][next];
            }
    unsigned long long g = __gcd(dp[n][0], p6[n]);
    if (dp[n][0] % p6[n] == 0)
        cout << dp[n][0] / p6[n] << endl;
    else
        cout << dp[n][0] / g << "/" << p6[n] / g << endl;
}
int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int tc;
    cin >> tc;
    for (int i = 0; i < tc; ++i) {
        cout << "Case " << i + 1 << ": ";
        solve();
    }
}
```

1068 - Investigation

Time Limit: 2 second(s)

Memory Limit: 32 MB

An integer is divisible by 3 if the sum of its digits is also divisible by 3. For example, 3702 is divisible by 3 and 12 ($3+7+0+2$) is also divisible by 3. This property also holds for the integer 9.

In this problem, we will investigate this property for other integers.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case contains three positive integers **A**, **B** and **K** ($1 \leq A \leq B < 2^{31}$ and $0 < K < 10000$).

Output

For each case, output the case number and the number of integers in the range [**A**, **B**] which are divisible by **K** and the sum of its digits is also divisible by **K**.

Sample Input	Output for Sample Input
3	Case 1: 20
1 20 1	Case 2: 5
1 20 2	Case 3: 64
1 1000 4	

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
string to_s(int t) {
    stringstream ss;
    ss << t;
    return ss.str();
}
string target;
int k;
int dp[10][2][100][100];
int go(int i, int top, int mod_sum, int mod_prod) {
    if (i == target.size()) {
        return (mod_sum == 0) && (mod_prod == 0);
    }
    if (dp[i][top][mod_sum][mod_prod] != -1)
        return dp[i][top][mod_sum][mod_prod];
    int ans = 0;
    int mmax = top ? (target[i] - '0') : 9;
    for (int j = 0; j <= mmax; ++j) {
        ans += go(i + 1, top && (j == mmax), (mod_sum + j) % k, (((mod_prod * 10)
% k) + j) % k);
    }
    return dp[i][top][mod_sum][mod_prod] = ans;
}
void solve() {
    int a, b;
    cin >> a >> b >> k;
    a--;
    if (k > 99) {
        printf("0\n");
        return;
    }
    target = to_s(b);
    memset(dp, -1, sizeof dp);
    long long ans = go(0, 1, 0, 0);
    target = to_s(a);
    memset(dp, -1, sizeof dp);
    ans -= go(0, 1, 0, 0);
    printf("%d\n", ans);
}
int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL);
```



```
int tc;  
cin >> tc;  
for (int i = 0; i < tc; ++i) {  
    printf("Case %d: ", i + 1);  
    solve();  
}  
return 0;  
}
```

1071 - Baker Vai

Time Limit: 2 second(s)

Memory Limit: 32 MB

All of you must have heard the name of Baker Vai. Yes, he rides a bike and likes to help people. That's why he is popular amongst general people.

Baker Vai lives in a city which can be modeled as a 2D $m \times n$ matrix. Where the north-west corner is cell $1, 1$ and the south-east corner is cell m, n . In each cell there are certain amount of people who needs help which is already known to Baker Vai.

Each day Baker Vai starts his journey from the north-west corner and he can only go to east or south. This way he reaches the south-east corner of the city. After that he returns back to the north-west, but this time he can only move to west or north. He doesn't want a cell to be visited twice other than the two corners. And if he visits a cell, he helps all the people in the cell.

Now you are given the map of the city and the number of people who need help in all cells for a particular day. You have to help Baker Vai finding the maximum number of people he can help in that day.

Input

Input starts with an integer T (≤ 25), denoting the number of test cases.

Each case contains a blank line and two integers, m, n ($2 \leq m, n \leq 100$). Each of the next m lines will contain n integers, denoting the number of people who are in need. In a cell there will be no more than 20 people and a cell can be empty, too.

Output

For each test case, print the case number and the maximum number of people Baker Vai can help considering the above conditions.

Sample Input	Output for Sample Input
2 3 3 1 1 1 1 0 1 1 1 1 3 4 1 1 0 1 1 1 1 1 0 1 1 0 1	Case 1: 8 Case 2: 18

SOLUTION

```
using namespace std;
#include<algorithm>
#include<iostream>
#include<sstream>
#include<string>
#include<vector>
#include<queue>
#include<stack>
#include<map>
#include<climits>
#include<cstring>
#include<cstdio>
#include<cmath>
#define For(i,a) for(int i=0;i<a;++i)
#define foreach(x,v) for(typeof (v).begin() x = (v).begin(); x!= (v).end(); x++)
#define all(x) x.begin(),x.end()
#define rall(x) x.rbegin(),x.rend()
#define D(x) cout<< #x " = "<<(x)<<endl
#define Dbg if(1)
#define MAXNODES 1000
int data[105][105];
int dp[105][105][105];
int m,n;
int solve(){
    dp[1][1][2] = data[1][1] + data[1][2];
    for(int i=3;i<=n;++i) dp[1][1][i] = dp[1][1][i-1] + data[1][i];

    for(int i = 2; i<=m ;++i){
        for(int j=1;j<=n;++j)
            for(int k=j+1; k<=n;++k)
                dp[i][j][k] = dp[i-1][j][k] + data[i][j] + data[i][k];

        for(int j=1;j<n;++j)
            for(int k=j+1; k<=n;++k)
                dp[i][j][k] = max(dp[i][j][k], dp[i][j-1][k] + data[i][j]) ;

        for(int j=1;j<n-1;++j)
            for(int k=j+2; k<=n;++k)
                dp[i][j][k] = max(dp[i][j][k], dp[i][j][k-1] + data[i][k]) ;
    }
}
```

```

        return dp[m][n-1][n];
    }
    int main(){
        int numcas;cin>>numcas;
        for(int cid=1;cid<=numcas;++cid){
            cin>>m>>n;
            for(int i=0;i< m ;++i )
                for(int j =0;j< n;++j)
                    cin>>data[i+1][j+1];

            cout<<"Case "<<cid<<": "<<solve()<<endl;
        }
        return 0;
    }

```

1073 - DNA Sequence

Time Limit: 4 second(s)

Memory Limit: 32 MB

You are given a list of strings over the alphabet A (for adenine), C (cytosine), G (guanine), and T (thymine), and your task is to find the shortest string (which is typically not listed) that contains all given strings as substrings. If there are several such strings of shortest length, find the smallest in alphabetical/lexicographical order.

Input

Input starts with an integer **T** (≤ 35), denoting the number of test cases.

Each case starts with an integer denoting the number of strings **n** ($1 \leq n \leq 15$) in a single line. Then these **n** strings ($1 \leq \text{length} \leq 100$) follow, one on each line, and they consist of the letters 'A', 'C', 'G' and 'T' only.

Output

For each case, print the case number and the shortest (and lexicographically smallest) string according to the description above.

Sample Input	Output for Sample Input
2 2 TGCACA CAT 3 TAC ACT CTA	Case 1: TGCACAT Case 2: ACTAC

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
const int inf = 1 << 20;
char alphabet[] = "ACGT";
int target;
int ov[111][111];
int inter(vector<string> &words, int a, int b) {
    if (a == -1)
        return words[b].size();
    if (ov[a][b] != -1)
        return ov[a][b];
    for (int i = 0; i < words[a].size(); ++i) {
        bool suffix = true;
        for (int j = 0; j < words[b].size() && i + j < words[a].size() && suffix;
            ++j) {
            if (words[a][i + j] != words[b][j])
                suffix = false;
        }
        if (suffix) {
            int overlap = words[a].size() - i;
            return ov[a][b] = words[b].size() - overlap;
        }
    }
    return ov[a][b] = words[b].size();
}
int dp[111][(1 << 15) + 10];
int op[111][(1 << 15) + 10];
int go(vector<string> &words, int last, int seen) {
    if (seen == target)
        return dp[last + 1][seen] = 0;
    if (dp[last + 1][seen] != -1)
        return dp[last + 1][seen];
    int ans = inf;
    for (int i = 0; i < words.size(); ++i) if (!((seen >> i) & 1)) {
        ans = min(ans, go(words, i, seen | (1 << i)) + inter(words, last, i));
    }
    return dp[last + 1][seen] = ans;
}
string best;
void find_best(vector<string> &w, int last, int seen, string cur) {
    if (cur > best)
        return;
```

```

    if (seen == target) {
        if (cur < best)
            best = cur;
        return;
    }
    for (int i = 0; i < w.size(); ++i) if (!((seen >> i) & 1)) {
        int need = inter(w, last, i);
        if (dp[last + 1][seen] == (dp[i + 1][seen | (1 << i)] + need)) {
            int index = w[i].size() - need;
            find_best(w, i, seen | (1 << i), cur + w[i].substr(index, need));
        }
    }
}

void solve() {
    int n;
    cin >> n;
    vector<string> words(n);
    for (int i = 0; i < n; ++i) {
        cin >> words[i];
    }
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j) if (words[i].find(words[j]) != string::npos)
                words[j] = "";
    vector<string> cleaned;
    for (int i = 0; i < n; ++i)
        if (words[i] != "")
            cleaned.push_back(words[i]);
    memset(ov, -1, sizeof ov);
    memset(dp, -1, sizeof dp);
    memset(op, -1, sizeof op);
    target = (1 << cleaned.size()) - 1;
    go(cleaned, -1, 0);
    best = "Z";
    find_best(cleaned, -1, 0, "");
    cout << best << endl;
}

int main() {
    int tc;
    cin >> tc;
    for (int i = 0; i < tc; ++i) {
        printf("Case %d: ", i + 1);
        solve();
    }
    return 0;
}

```


1079 - Just another Robbery

Time Limit: 4 second(s)

Memory Limit: 32 MB

As Harry Potter series is over, Harry has no job. Since he wants to make quick money, (he wants everything quick!) so he decided to rob banks. He wants to make a calculated risk, and grab as much money as possible. But his friends - Hermione and Ron have decided upon a tolerable probability P of getting caught. They feel that he is safe enough if the banks he robs together give a probability less than P .

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case contains a real number P , the probability Harry needs to be below, and an integer N ($0 < N \leq 100$), the number of banks he has plans for. Then follow N lines, where line j gives an integer M_j ($0 < M_j \leq 100$) and a real number P_j . Bank j contains M_j millions, and the probability of getting caught from robbing it is P_j . A bank goes bankrupt if it is robbed, and you may assume that all probabilities are independent as the police have very low funds.

Output

For each case, print the case number and the maximum number of millions he can expect to get while the probability of getting caught is less than P .

Sample Input	Output for Sample Input
3 0.04 3 1 0.02 2 0.03 3 0.05 0.06 3 2 0.03 2 0.03 3 0.05 0.10 3 1 0.03 2 0.02 3 0.05	Case 1: 2 Case 2: 4 Case 3: 6

Note

For the first case, if he wants to rob bank 1 and 2, then the probability of getting caught is $0.02 + (1 - 0.02) * .03 = 0.0494$ which is greater than the given probability (0.04). That's why he has only option, just to rob rank 2.

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <map>
#include <string.h>
using namespace std;
double p[110];
int million[110];
int n;
long long count;
double tol;
int main()
{
    int t;
    int sum;
    int ans;
    double val[10005];
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%lf", &tol);
        scanf("%d", &n);
        sum = 0;
        for (int i = 0; i < n; i++) {
            scanf("%d", &million[i]);
            scanf("%lf", &p[i]);
            sum += million[i];
        }
        for (int i = 1; i <= sum; i++) {
            val[i] = 1;
        }
        val[0] = 0;
        int index;
        for (int j = 0; j < n; j++) {
            for(int i = sum; i >= 1; i--) {

                index = i - million[j];
                if(index >= 0) {
                    val[i] = min(val[i], val[index] + (1 -
val[index])) * p[j]);
                }
            }
        }
        for (int i = 0; i <= sum; i++) {
            if(val[i] < tol) {
```

```
        ans = i;
    }
}
printf("Case %d: %d\n", cs, ans);
}
}
```

1084 - Winter

Time Limit: 2 second(s)

Memory Limit: 32 MB

Winter is coming. In a land far away, N men are spending the nights in a valley in a largest field. The valley is so narrow that it can be considered to be a straight line running east-to-west.

Although standing in the valley does shield them from the wind, the group still shivers during the cold nights. They, like anyone else, would like to gather together for warmth.

Near the end of each day, each man i finds himself somewhere in the valley at a unique location L_i . The men want to gather into groups of three or more persons since two persons just aren't warm enough. They want to be in groups before sunset, so the distance K each man can walk to form a group is limited. Determine the smallest number of groups the men can form.

Input

Input starts with an integer T (≤ 15), denoting the number of test cases.

Each case starts with two integers N ($1 \leq N \leq 10^5$) and K ($1 \leq K \leq 10^6$). Each of the next N line contains an integer L_i ($1 \leq L_i \leq 10^8$).

Output

For each case, print the case number and smallest number of groups the men can gather into. If there is no way for all the men to gather into groups of at least size three, output -1.

Sample Input	Output for Sample Input
2 6 10 2 10 15 13 28 9 3 1 1 10 20	Case 1: 2 Case 2: -1

Note

Dataset is huge, use faster I/O methods.

SOLUTION

```
#include <stdio.h>
#include <iostream>
#include <limits.h>
#include <stdlib.h>
#include <algorithm>
#include <queue>
#include <string.h>
using namespace std;
long long a[100005];
bool vis[100005];
int n;
long long k;
int bfs()
{
    pair <int, int> temp;
    temp.first = 0;
    temp.second = 0;
    queue < pair <int, int> > q;
    q.push(temp);
    vis[0] = 1;
    while(!q.empty()) {
        int i = q.front().first;
        int count = q.front().second;
        q.pop();

        int j = i;
        int p;
        p = 0;
        if(j == n) {
            return count;
        }
        while((a[j] - a[i]) <= 2 * k and j < n) {
            j++;
            p++;
        }
        int k;
        k = 0;
        while((p - k) >= 3) {
            if(vis[j - k] == 0) {
                q.push(make_pair(j - k, count + 1));
                vis[j-k] = 1;
            }
            k++;
        }
    }
}
```

```

        }
    }
    return -1;
}
int main()
{
    int t;

    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        scanf("%lld", &k);

        memset(vis, 0, sizeof vis);
        for (int i = 0; i < n; i++) {
            scanf("%lld", &a[i]);
        }
        sort(a, a + n);

        printf("Case %d: %d\n", cs, bfs());
    }
}

```

1086 - Jogging Trails

Time Limit: 2 second(s)

Memory Limit: 32 MB

Robin is training for a marathon. Behind his house is a park with a large network of jogging trails connecting water stations. Robin wants to find the shortest jogging route that travels along every trail at least once.

For each case, there should be one line of output giving the length of Robin's jogging route.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case contains two positive integers n ($2 \leq n \leq 15$), the number of water stations, and m ($0 \leq m \leq 1000$), the number of trails. For each trail, there is one subsequent line of input containing three positive integers: the first two, between 1 and n , indicating the water stations at the end points of the trail; the third indicates the length of the trail, in cubits. There may be more than one trail between any two stations; each different trail is given only once in the input; each trail can be travelled in either direction. It is possible to reach any trail from any other trail by visiting a sequence of water stations connected by trails. Robin's route may start at any water station, and must end at the same station.

Output

For each case, print the case number and the minimum possible length of Robin's jogging route

Sample Input	Output for Sample Input
1 4 5 1 2 3 2 3 4 3 4 5 1 4 10 1 3 12	Case 1: 41

SOLUTION

```
//The great chinese postman problem
#include <queue>
#include <vector>
#include <stdlib.h>
#include <iostream>
#include <limits.h>
#include <algorithm>
#include <string.h>
#include <stdio.h>
#define set(x, n) (x ^ (1 << n))
#define check(x, n) (x & (1 << n))
using namespace std;
int n;
int ind;
int edge_count[1028];
long long floyd[1028][1028];
int ans;
long long dp[1 << 17];
int floyd_warshal()
{
    long long t;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if(floyd[i][j] == 0) {
                floyd[i][j] = INT_MAX;
            }
        }
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if(floyd[i][k] != INT_MAX and floyd[k][j] !=
INT_MAX) {
                    t = floyd[i][k] + floyd[k][j];
                    if(t < floyd[i][j]) {
                        floyd[i][j] = t;
                    }
                }
            }
        }
    }
}
```

```

long long chinese(int bit)
{

    int x;
    long long mini;
    int i;

    mini = INT_MAX;

    if(bit == 0) {
        return 0;
    }

    if(dp[bit] != -1) {
        return dp[bit];
    }
    for(i = 1; i <= n; i++) {
        if(check(bit, i)) {
            break;
        }
    }
    for (int j = i + 1; j <= n; j++) {
        int temp = bit;
        if(check(bit, i) and check(bit, j )) {
            temp = set(temp, i);
            temp = set(temp, j);
            mini = min(mini, floyd[i][j] + chinese(temp));
        }
    }

    return dp[bit] = mini;
}

int travel(int cs)
{
    int x;
    int y;
    int start;
    long long mini;
    long long w;
    int m;
    int bit;

    memset(edge_count, 0, sizeof edge_count);
    memset(dp, -1, sizeof dp);

```

```

ans = 0;
bit = 0;
cin >> n; //no of joints
cin >> m; // no of streets
for (int i = 0; i <= n; i++) {
    for (int j = 0; j <= n; j++) {
        floyd[i][j] = INT_MAX;
    }
}

for (int i = 0; i < m; i++) {

    cin >> x; // street starting point
    cin >> y; // street ending point
    cin >> w; // street length
    ans += w;

    floyd[x][y] = min(w, floyd[x][y]);
    floyd[y][x] = min(w, floyd[y][x]);

    edge_count[x]++;
    edge_count[y]++;

}

floyd_warshal(); // calculate the shortest path between all the joints

for (int i = 1; i <= n; i++) {
    if(edge_count[i] % 2 == 1) {
        bit = set(bit, i); // mark joints with odd degree
    }
}
ans += chinese(bit); // try all combination of odd vertices and choose
the combination with minimum distance
printf("Case %d: %d\n", cs, ans);
}

int main()
{
    int t;
    cin >> t; // no of test cases
    for (int cs = 1; cs <= t; cs++) {
        travel(cs);
    }
}

```


1092 - Lighted Panels

Time Limit: 3 second(s)

Memory Limit: 32 MB

You are given an $R \times C$ 2D grid consisting of several light panels. Each cell contains either a '*' or a '.'. '*' means the panel is on, and '.' means it's off. If you touch a panel, its state will be toggled. That means, if you touch a panel that's on, it will turn off, and if you touch a panel that's off, it will turn on. But if we touch a panel, all its horizontal, vertical, and diagonal **adjacent** panels will also toggle their states.

Now you are given the configuration of the grid. Your goal is to turn on all the lights. Print the minimum number of touches required to achieve this goal.

Input

Input starts with an integer T (≤ 125), denoting the number of test cases.

Each test case starts with two integers R ($1 \leq R \leq 8$) and C ($1 \leq C \leq 8$). Then there will be R lines each containing C characters ('*' or '.').

Output

For each test case, print the case number and the minimum number of touches required to have all the light panels in the board on at the same time. If it is not possible then print "impossible".

Sample Input	Output for Sample Input
4 5 5 ***** *...* *...* *...* ***** 1 2 .* 3 3 **. **. ... 4 4 *... **.. ..** ...*	Case 1: 1 Case 2: impossible Case 3: 2 Case 4: 10

SOLUTION

```
#include <stdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int INF = 0x3f3f3f3f;
int R, C;
int dr[] = {-1, 0, 0, 1, -1, 1, -1, 1};
int dc[] = {0, -1, 1, 0, -1, -1, 1, 1};
bool grid[8][8], orig[8][8];
inline bool valid(int r, int c) {
    return 0 <= r && r < R && 0 <= c && c < C;
}
inline void toggle(int r, int c) {
    int i, r1, c1;
    grid[r][c] ^= true;
    for(i = 0; i < 8; i++) {
        r1 = r + dr[i], c1 = c + dc[i];
        if(valid(r1, c1)) grid[r1][c1] ^= true;
    }
}
int main() {
    int test, cs, i, j, best, moves, r, c, done;
    char buff[9];
    scanf("%d", &test);
    for(cs = 1; cs <= test; cs++) {
        scanf("%d %d", &R, &C);
        for(i = 0; i < R; i++) {
            scanf("%s", buff);
            for(j = 0; j < C; j++) grid[i][j] = (buff[j] == '*');
        }
        best = INF;
        memcpy(orig, grid, sizeof grid);
        for(i = 0; i < (1 << C); i++) {
            for(j = 0; j < (1 << R); j++) {
                moves = 0;
                memcpy(grid, orig, sizeof orig);
                for(c = 0; c < C; c++) {
                    if(i & (1 << c)) {
                        toggle(0, c);
                        moves++;
                    }
                }
            }
            for(r = 0; r < R; r++) {
```

```

        if(j & (1 << r)) {
            toggle(r, 0);
            moves++;
        }
    }
    for(r = 1; r < R; r++) {
        for(c = 1; c < C; c++) {
            if(!grid[r-1][c-1]) {
                toggle(r, c);
                moves++;
            }
        }
    }
    done = 0;
    for(r = 0; r < R && !done; r++)
        for(c = 0; c < C && !done; c++)
            if(!grid[r][c]) done = true;
    if(!done) best = min(best, moves);
}

}
if(best == INF) printf("Case %d: impossible\n", cs);
else printf("Case %d: %d\n", cs, best);
}
return 0;
}

```

1095 - Arrange the Numbers

Time Limit: 2 second(s)

Memory Limit: 32 MB

Consider this sequence $\{1, 2, 3 \dots N\}$, as an initial sequence of first N natural numbers. You can rearrange this sequence in many ways. There will be a total of $N!$ arrangements. You have to calculate the number of arrangement of first N natural numbers, where in first M positions; exactly K numbers are in their initial position.

For Example, $N = 5, M = 3, K = 2$

You should count this arrangement $\{1, 4, 3, 2, 5\}$, here in first 3 positions 1 is in 1st position and 3 in 3rd position. So exactly 2 of its first 3 are in there initial position.

But you should not count $\{1, 2, 3, 4, 5\}$.

Input

Input starts with an integer T (≤ 1000), denoting the number of test cases.

Each case contains three integers N ($1 \leq N \leq 1000$), M ($M \leq N$), K ($0 < K \leq M$).

Output

For each case, print the case number and the total number of possible arrangements modulo 1000000007.

Sample Input	Output for Sample Input
2 5 3 2 10 6 3	Case 1: 12 Case 2: 64320

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
const long long mod = 1000000007;
const int MN = 1000 + 10;
long long fac[MN];
long long dp[MN][MN];
long long comb(int n, int k) {
    if (k == 0 || n == k) return 1;
    if (dp[n][k] == -1)
        dp[n][k] = (comb(n - 1, k) + comb(n - 1, k - 1)) % mod;
    return dp[n][k];
}
// finds how many permutations of a size-n set does not have any of the first
k elements
// in their correspondent position (kth position).
long long dea(int n, int k) {
    long long ans = fac[n];
    for (int i = 1; i <= k; ++i) {
        ans -= ((i & 1) ? 1 : -1) * (comb(k, i) * fac[n - i]) % mod;
        ans = (ans + mod) % mod;
    }
    return ans;
}
void solve() {
    int n, m, k;
    cin >> n >> m >> k;
    long long ans = (comb(m, k) * dea(n - k, m - k)) % mod;
    printf("%lld\n", ans);
}
int main() {
    ios_base::sync_with_stdio(false);
    int tc; cin >> tc;
    fac[0] = 1;
    for (int i = 1; i < MN; ++i)
        fac[i] = (fac[i - 1] * i) % mod;
    memset(dp, -1, sizeof dp);
    for (int i = 0; i < tc; ++i) {
        printf("Case %d: ", i + 1);
        solve();
    }
    return 0;
}
```


1105 - Fi Binary Number

Time Limit: 2 second(s)

Memory Limit: 32 MB

A Fi-binary number is a number that contains only 0 and 1. It does not contain any leading 0. And also it does not contain 2 consecutive 1. The first few such number are 1, 10, 100, 101, 1000, 1001, 1010, 10000, 10001, 10010, 10100, 10101 and so on. You are given n . You have to calculate the n^{th} Fi-Binary number.

Input

Input starts with an integer T (≤ 10000), denoting the number of test cases.

Each case contains an integer n ($1 \leq n \leq 10^9$).

Output

For each case, print the case number and the n^{th} Fi-Binary number

Sample Input	Output for Sample Input
4	Case 1: 10010
10	Case 2: 101010
20	Case 3: 1010001
30	Case 4: 10001001
40	

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
int a[100];
int dp[100];
int maxi;
int n;
int explore(int sum)
{
    if(sum <= 0) {

        for (int j = maxi; j >= 0; j--) {
            printf("%d", a[j]);
        }
        printf("\n");
        return 0;
    }
    if(sum == 1) {
        a[0] = 1;
        maxi = max(maxi, 0);
        explore(0);
        return 0;
    }
    for (int i = 0; 1; i++) {
        if(dp[i+1] >= sum and dp[i] < sum) {
            a[i+1] = 1;
            sum = sum - (dp[i]+1);
            maxi = max(maxi, i+1);
            explore(sum);
            break;
        }
    }
}

int main()
{
    int j;
    int k;
    int t;
    int temp;
    j = 1;
    k = 2;
    dp[0] = 1;
```

```
for (int i = 1; i < 44; i++) {  
    dp[i] = dp[i-1] + j;  
    temp = k;  
    k = j + k;  
    j = temp;  
  
}  
  
scanf("%d", &t);  
for (int cs = 1; cs <= t; cs++) {  
    scanf("%d", &n);  
    memset(a, 0, sizeof(a));  
    maxi = -1;  
    printf("Case %d: ", cs);  
    explore(n);  
}  
}
```

1106 - Gone Fishing

Time Limit: 2 second(s)

Memory Limit: 32 MB

John is going on a fishing trip. He has h hours available, and there are n lakes in the area all reachable along a single, one-way road. John starts at lake 1, but he can finish at any lake he wants. He can only travel from one lake to the next one, but he does not have to stop at any lake unless he wishes to. For each i (1 to $n-1$), the number of 5-minute intervals it takes to travel from lake i to lake $i+1$ is denoted t_i . For example, $t_3=4$ means that it takes 20 minutes to travel from lake 3 to 4.

To help plan his fishing trip, John has gathered some information about the lakes. For each lake i , the number of fish expected to be caught in the initial 5 minutes, denoted f_i , is known. Each 5 minutes of fishing decreases the number of fish expected to be caught in the next 5-minute interval by a constant rate of d_i . If the number of fish expected to be caught in an interval is less than or equal to d_i , there will be no more fish left in the lake in the next interval. To simplify the planning, John assumes that no one else will be fishing at the lakes to affect the number of fish he expects to catch. Write a program to help John plan his fishing trip to maximize the number of fish expected to be caught. The number of minutes spent at each lake must be a multiple of 5.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing two integers n ($2 \leq n \leq 25$) and h ($1 \leq h \leq 16$). Next, there is a line of n integers specifying f_i ($0 \leq f_i \leq 1000$), then a line of n integers d_i ($0 \leq d_i \leq 1000$), and finally, a line of $n-1$ integers denoting t_i ($0 < t_i < 192$).

Output

For each test case, print the case number first. Then print the number of minutes spent at each lake, separated by commas, for the plan achieving the maximum number of fish expected to be caught. This is followed by a line containing the number of fish expected. If multiple plans exist, choose the one that spends as long as possible at lake 1. If there is still a tie, choose the one that spends as long as possible at lake 2, and so on.

Sample Input	Output for Sample Input
3 2 1 10 1 2 5 2 4 4 10 15 20 17	Case 1: 45, 5 Number of fish expected: 31 Case 2: 240, 0, 0, 0 Number of fish expected: 480 Case 3:

0 3 4 3	115, 10, 50, 35
1 2 3	Number of fish expected: 724
4 4	
10 15 50 30	
0 3 4 3	
1 2 3	

SOLUTION

```
#include<bits/stdc++.h>
#define pause system("pause");
#define FOR(s,e,inc) for(int i=s;i<=e;i+=inc)
#define mod 1000000007
#define inf 1<<30
#define pb push_back
#define ppb pop_back
#define mp make_pair
#define F first
#define S second
#define sz(x) ((int)x.size())
#define sqr(x) ( (x)* (x) )
#define eps 1e-9
#define gcd(x,y) __gcd(x,y)
#define on(x,w)  x=x|(1<<w)
#define check(x,w) (x&(1<<w))
#define all(x) (x).begin(),(x).end()
#define pf printf
#define reset(x,v) memset(x,v,sizeof(x));
#define AND &&
#define OR ||
typedef long long ll;
typedef unsigned long long ull;
using namespace std;
template<class T>
inline T mod_v(T num)
{
    if(num>=0)
        return num%mod;
    else
        return (num%mod+mod)%mod;
}
template<class T>
inline void memset1(vector<T>&v,T value)
{
    for(int i=0;i<sz(v);i++)
        v[i]=value;
}
template<class T>
inline void memset2(vector<vector<T> >&v,T value)
{
    for(int i=0;i<sz(v) ;i++)
        for(int j=0;j<sz(v[i]);j++)
```



```

        v[i][j]=value;
    }
    template<class T>
    T fast_pow(T n , T p)
    {
        if(p==0) return 1;
        if(p%2)
        {
            T g=mod_v ( mod_v(n) * mod_v(fast_pow(n,p-1)) );
            return g;
        }
        else
        {
            T g=fast_pow(n,p/2);
            g=mod_v( mod_v(g) * mod_v(g) ) ;
            return g;
        }
    }
    template<class T>
    inline T modInverse(T n)
    {
        return fast_pow(n,mod-2);
    }
    template<class T>
    inline void debug(string S1,T S2,string S3)
    {
        cout<<S1<<S2<<S3;
    }
    template<class T>
    inline T in()
    {
        register char c=0;
        register T num=0;
        bool n=false;
        while(c<33)c=getchar();
        while(c>33){
            if(c=='-')
                n=true;
            else num=num*10+c-'0';
            c=getchar();
        }
        return n?-num:num;
    }
#ifdef ONLINE_JUDGE
#   define p(x) cout<<x<<endl;
#else if

```

```

# define print(x) 0;
#endif
//.....Code start from here ! .....
int dp[26][16*12+5];
int n,h,mx;
int f[26];
int t[26];
int d[26];
int re(int pos,int minute)
{
    if(pos>=n || minute<0) return 0;
    int &r=dp[pos][minute];
    if(r!=-1) return r;
    r=0;
    int fish=0,tmp;
    r=max(r,fish+re(pos+1,minute-t[pos] ) );
    for(int i=1,j=0;i<=minute;i++,j++)
    {
        if(f[pos]-d[pos]*j>0)
        {
            fish+=(f[pos]-d[pos]*j);
        }
        r=max(r,fish+re(pos+1,minute-i-t[pos] ) );
    }
    return r;
}

void print_sol(int pos,int minute,int cfish)
{
    if(pos>=n ) return ;
    int fish=0;
    if(cfish==0)
    {
        if(pos+1==n)
            printf("0\n");
        else
        {
            printf("0, ");
            print_sol(pos+1,minute-t[pos],cfish);
        }
        return ;
    }
    for(int i=minute;i>=0;i--)
    {
        fish=0;
        for(int j=1,k=0;j<=i;j++,k++)
        {

```

```

        if(f[pos]-d[pos]*k>0)
        {
            fish+=(f[pos]-d[pos]*k);
        }
    }
    //printf("Hi pos=%d %d %d
%d\n",pos,fish,dp[pos+1][minute-i-t[pos]],i*5);
    if(cfish==fish+re(pos+1,minute-i-t[pos])) )
    {
        if(pos+1==n)
            printf("%d\n",i*5);
        else
        {
            printf("%d, ",i*5);
            print_sol(pos+1,minute-i-t[pos],cfish-fish);
        }
        break;
    }
}
}
int main()
{
    int te,tcase=1;
    te=in<int>();
    while(te--)
    {
        n=in<int>();
        h=in<int>();
        h=h*12;
        for(int i=0;i<n;i++)
            f[i]=in<int>();
        for(int i=0;i<n;i++)
            d[i]=in<int>();
        for(int i=0;i<n-1;i++)
            t[i]=in<int>();
        printf("Case %d:\n",tcase++);
        reset(dp,-1);
        mx=re(0,h);
        print_sol(0,h,mx);
        printf("Number of fish expected: %d\n",mx);
    }
    return 0;
}

```

1110 - An Easy LCS

Time Limit: 2 second(s)

Memory Limit: 32 MB

LCS means 'Longest Common Subsequence' that means two non-empty strings are given; you have to find the Longest Common Subsequence between them. Since there can be many solutions, you have to print the one which is the lexicographically smallest. Lexicographical order means dictionary order. For example, 'abc' comes before 'abd' but 'aaz' comes before 'abc'.

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case starts with a blank line. The next two lines will contain two strings of length **1** to **100**. The strings contain lowercase English characters only.

Output

For each case, print the case number and the lexicographically smallest LCS. If the LCS length is **0** then just print ':('.

Sample Input	Output for Sample Input
3 ab ba zxcvbn hjgasbznxbzmx you kjhs	Case 1: a Case 2: zxb Case 3: :(

SOLUTION

```
#include <iostream>
#include <stdio.h>
using namespace std;
string minim(string x, string y)
{
    if(x.length() < y.length()) {
        return y;
    }

    if(x.length() > y.length()) {
        return x;
    }
    for (int i = 0; i < x.length(); i++) {
        if(x[i] < y[i]) {
            return x;
        }
        if(x[i] > y[i]) {
            return y;
        }
    }
    return x;
}

int main()
{
    int t;
    string x;
    string y;
    string temp;
    char xx[110];
    char yy[110];
    char ans[110];
    int n;
    int m;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {

        string a[110][110];
        scanf("%s", xx);
        scanf("%s", yy);

        x = xx;
        y = yy;
```

```

n = x.length();
m = y.length();
for (int i = n; i >= 0; i--) {
    for (int j = m; j >= 0; j--) {
        if(i == n or j == m) {
            a[i][j] = "";
            continue;
        }
        if(x[i] == y[j]) {
            a[i][j] = x[i] + a[i+1][j+1];
            continue;
        }
        a[i][j] = minim(a[i+1][j],a[i][j+1]);
    }
}
for (int i = 0; i < a[0][0].length(); i++) {
    ans[i] = a[0][0][i];
}
ans[a[0][0].length()] = '\0';
if(a[0][0].length() != 0) {
    printf("Case %d: %s\n", cs, ans);
}
else {
    printf("Case %d: :(\n", cs);
}
}
}
}

```

1119 - Pimp My Ride

Time Limit: 2 second(s)

Memory Limit: 32 MB

Today, there are quite a few cars, motorcycles, trucks and other vehicles out there on the streets that would seriously need some refurbishment. You have taken on this job, ripping off a few dollars from a major TV station along the way. Of course, there's a lot of work to do, and you have decided that it's getting too much. Therefore you want to have the various jobs like painting, interior decoration and so on done by garages. Unfortunately, those garages are very specialized, so you need different garages for different jobs. More so, they tend to charge you the more the better the overall appearance of the car is. That is, a painter might charge more for a car whose interior is all leather. As those "surcharges" depend on what job is done and which jobs have been done before, you are currently trying to save money by finding an optimal order for those jobs.

Individual jobs are numbered **1** through **n**. Given the base price **p** for each job and a surcharge **s** for every pair of jobs **(i, j)**, meaning that you have to pay additional **s** for job **i**, if and only if job **j** was completed before, you are to compute the minimum total costs needed to finish all jobs.

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case starts with an integer **n** ($1 \leq n \leq 14$) denoting number of jobs. Then follow **n** lines, each containing exactly **n** integers. The **i**th line contains the surcharges that have to be paid in garage number **i** for the **i**th job and the base price for job **i**. More precisely, on the **i**th line, the **i**th integer is the base price for job **i** and the **j**th integer $i \neq j$ is the surcharge for job **i** that applies if job **j** has been done before. The prices will be non-negative integers smaller than or equal to **100000**.

Output

For each case, print the case number and the minimum total cost.

Sample Input	Output for Sample Input
2 2 10 10 9000 10 3 14 23 0 0 14 0 1000 9500 14	Case 1: 30 Case 2: 42

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <limits.h>
#include <string.h>
using namespace std;
#define set(x, i) (x | (1 << i))
#define check(x, i) (x & (1 << i))
int a[20][20];
long long dp[20][2 << 16];
int n;
int explore(int i, int x)
{
    int ok;
    int sum;
    int mini = INT_MAX;
    sum = 0;
    ok = 0;
    if(dp[i][x] != -1) {
        return dp[i][x];
    }
    for (int j = 1; j <= n; j++) {
        if(check(x, j)) {
            sum += a[i][j];
        }
        else {
            ok = 1;
        }
    }
    if(ok == 0) {
        return sum;
    }
    for (int j = 1; j <= n; j++) {
        if(check(x, j) == 0) {
            mini = min(mini, explore(j, set(x, j)));
        }
    }
    mini = mini + sum;
    return dp[i][x] = mini;
}

int main()
{
    int t;
```

```
scanf("%d", &t);
for (int cs = 1; cs <= t; cs++) {
    scanf("%d", &n);
    memset(a, 0, sizeof a);
    memset(dp, -1, sizeof dp);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    printf("Case %d: %d\n", cs, explore(0, 0));
}
}
```

1122 - Digit Count

Time Limit: 2 second(s)

Memory Limit: 32 MB

Given a set of digits **S**, and an integer **n**, you have to find how many **n**-digit integers are there, which contain digits that belong to **S** and the difference between any two adjacent digits is not more than two.

Input

Input starts with an integer **T** (≤ 300), denoting the number of test cases.

Each case contains two integers, **m** ($1 \leq m < 10$) and **n** ($1 \leq n \leq 10$). The next line will contain **m** integers (**from 1 to 9**) separated by spaces. These integers form the set **S** as described above. These integers will be distinct and given in ascending order.

Output

For each case, print the case number and the number of valid **n**-digit integers in a single line.

Sample Input	Output for Sample Input
3	Case 1: 5
3 2	Case 2: 9
1 3 6	Case 3: 9
3 2	
1 2 3	
3 3	
1 4 6	

Note

For the first case the valid integers are

11
13
31
33
66

SOLUTION

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
using namespace std;
int m;
int n;
int a[12];
long long ans[12][11];
long long getCount(int i, int x)
{
    long long count;

    count = 0;

    if(ans[i][x] != -1) {
        return ans[i][x];
    }

    if(i == n) {
        return 1;
    }
    for (int j = 0; j < m; j++) {
        if(x == 0 or (abs(a[j] - x) <= 2)) {
            count += getCount(i+1, a[j]);
        }
    }
    ans[i][x] = count;
    return count;
}
int main()
{
    int t;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        memset(ans, -1, sizeof(ans));
        scanf("%d", &m);
        scanf("%d", &n);
        for (int i = 0; i < m; i++) {
            scanf("%d", &a[i]);
        }
        printf("Case %d: %lld\n",cs, getCount(0, 0));
    }
}
```


1125 - Divisible Group Sums

Time Limit: 2 second(s)

Memory Limit: 32 MB

Given a list of N numbers you will be allowed to choose any M of them. So you can choose in ${}^N C_M$ ways. You will have to determine how many of these chosen groups have a sum, which is divisible by D .

Input

Input starts with an integer T (≤ 20), denoting the number of test cases.

The first line of each case contains two integers N ($0 < N \leq 200$) and Q ($0 < Q \leq 10$). Here N indicates how many numbers are there and Q is the total number of queries. Each of the next N lines contains one 32 bit signed integer. The queries will have to be answered based on these N numbers. Each of the next Q lines contains two integers D ($0 < D \leq 20$) and M ($0 < M \leq 10$).

Output

For each case, print the case number in a line. Then for each query, print the number of desired groups in a single line.

Sample Input	Output for Sample Input
2 10 2 1 2 3 4 5 6 7 8 9 10 5 1 5 2 5 1 2 3 4 5 6 6 2	Case 1 : 2 9 Case 2 : 1

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
const int MN = 201,
        MD = 21,
        MM = 11;
long long dp[MN][MM][MD];
int D;
int sum(int a, int b) {
    long long ans = a + b;
    ans %= D;
    if (ans < 0)
        ans += D;
    return ans;
}
long long go(const vector<int> &a, int i, int t, int acc) {
    if (i == a.size() || t == 0)
        return (t == 0) && (acc % D) == 0;
    if (dp[i][t][acc] != -1)
        return dp[i][t][acc];
    return dp[i][t][acc] = go(a, i + 1, t, acc) + go(a, i + 1, t - 1, sum(acc,
a[i]));
}
void solve() {
    int n, q;
    cin >> n >> q;
    vector<int> a(n);
    for (int i = 0; i < n; ++i)
        cin >> a[i];
    int m, d;
    while (q--) {
        cin >> d >> m;
        D = d;
        memset(dp, -1, sizeof dp);
        vector<int> b = a;
        for (int i = 0; i < a.size(); ++i) {
            b[i] %= d;
            if (b[i] < 0)
                b[i] += d;
        }
        printf("%lld\n", go(b, 0, m, 0));
    }
}
int main() {
```

```
ios_base::sync_with_stdio(false);cin.tie(NULL);  
int tc; cin >> tc;  
for (int i = 0; i < tc; ++i) {  
    printf("Case %d:\n", i + 1);  
    solve();  
}  
return 0;  
}
```


1126 - Building Twin Towers

Time Limit: 4 second(s)

Memory Limit: 32 MB

Professor Sofdor Ali is fascinated about twin towers. So, in this problem you are working as his assistant, and you have to help him making a large twin towers.

For this reason he gave you some rectangular bricks. You can pick some of the bricks and can put bricks on top of each other to build a tower. As the name says, you want to make two towers that have equal heights. And of course the height of the towers should be positive.

For example, suppose there are three bricks of heights 3, 4 and 7. So you can build two towers of height 7. One contains a single brick of height 7, and the other contains a brick of height 3 and a brick of height 4. If you are given bricks of heights 2, 2, 3 and 4 then you can make two towers with height 4 (just don't use brick with height 3).

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case starts with an integer **n** ($1 \leq n \leq 50$), denoting the number of bricks. The next line contains **n** space separated integers, each containing the height of the brick **h_i** ($1 \leq h_i \leq 500000$). You can safely assume that the sum of heights of all bricks will not be greater than **500000**.

Output

For each case, print the case number and the height of the tallest twin towers that can be built. If it's impossible to build the twin towers as stated, print **"impossible"**.

Sample Input	Output for Sample Input
4	Case 1: 7
3	Case 2: impossible
3 4 7	Case 3: 21
3	Case 4: 64
10 9 2	
2	
21 21	
9	
15 15 14 24 14 3 20 23 15	

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
const int MS = 250000;
const int MN = 2 * MS + 1;
const int inf = MS * 10;
int dp[2][MN];
int go(vector<int> &a, int m) {
    int cur = 0;
    for (int i = -MS; i <= MS; ++i) {
        dp[cur][MS + i] = -inf;
        dp[cur][MS + i] = -inf;
    }
    dp[cur][MS] = 0;
    for (int i = a.size() - 1; i >= 0; --i) {
        cur ^= 1;
        for (int acc = -m; acc <= m; ++acc) {
            int best = -inf;
            if (acc + a[i] <= m) best = max(best, dp[cur ^ 1][MS + acc + a[i]] +
a[i]);
            if (abs(acc - a[i]) <= m) best = max(best, dp[cur ^ 1][MS + acc - a[i]]
+ a[i]);
            best = max(best, dp[cur ^ 1][MS + acc]);
            dp[cur][MS + acc] = best;
        }
    }
    return dp[cur][MS];
}
void solve() {
    int n; cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; ++i)
        cin >> a[i];
    int m = accumulate(a.begin(), a.end(), 0);
    int ans = go(a, (m + 1) / 2);
    if (ans <= 0)
        puts("impossible");
    else
        printf("%d\n", ans / 2);
}
int main() {
    // ios_base::sync_with_stdio(false);cin.tie(NULL);
    int tc; cin >> tc;
    for (int i = 0; i < tc; ++i) {
```

```
    printf("Case %d: ", i + 1);  
    solve();  
}  
return 0;  
}
```

1134 - Be Efficient

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are given an array with **N** integers, and another integer **M**. You have to find the number of consecutive subsequences which are divisible by **M**.

For example, let **N** = 4, the array contains {2, 1, 4, 3} and **M** = 4.

The consecutive subsequences are {2}, {2 1}, {2 1 4}, {2 1 4 3}, {1}, {1 4}, {1 4 3}, {4}, {4 3} and {3}. Of these 10 'consecutive subsequences', only two of them adds up to a figure that is a multiple of 4 - {1 4 3} and {4}.

Input

Input starts with an integer **T** (≤ 10), denoting the number of test cases.

Each case contains two integers **N** ($1 \leq N \leq 10^5$) and **M** ($1 \leq M \leq 10^5$). The next line contains **N** space separated integers forming the array. Each of these integers will lie in the range $[1, 10^5]$.

Output

For each case, print the case number and the total number of consecutive subsequences that are divisible by **M**.

Sample Input	Output for Sample Input
2 4 4 2 1 4 3 6 3 1 2 3 4 5 6	Case 1: 2 Case 2: 11

Note

Dataset is huge. Use faster i/o methods.

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
long long fact_cal(int x)
{
    long long nplus1 = x;
    nplus1++;
    return (long long) (x * nplus1) / 2;
}
int main()
{
    int t;
    int n;
    int m;
    int a[500004];
    int count[500005];
    long long sum;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        scanf("%d", &m);
        memset(count, 0, sizeof(count));
        sum = 0;
        for (int i = 0; i < n; i++) {
            scanf("%d", &a[i]);
        }

        a[0] = a[0] % m;
        for (int i = 1; i < n; i++) {
            a[i] = (a[i-1] + a[i]) % m;
        }

        for (int i = 0; i < n; i++) {
            count[a[i]]++;
        }
        sum += count[0];
        long long temp;

        for (int i = 0; i <= 500000; i++) {
            sum = sum + fact_cal(count[i] - 1);
        }
        printf("Case %d: %lld\n", cs, sum);
    }
}
```


1140 - How Many Zeroes?

Time Limit: 2 second(s)

Memory Limit: 32 MB

Jimmy writes down the decimal representations of all natural numbers between and including m and n , ($m \leq n$). How many zeroes will he write down?

Input

Input starts with an integer T (≤ 11000), denoting the number of test cases.

Each case contains two unsigned 32-bit integers m and n , ($m \leq n$).

Output

For each case, print the case number and the number of zeroes written down by Jimmy.

Sample Input	Output for Sample Input
5	Case 1: 1
10 11	Case 2: 22
100 200	Case 3: 92
0 500	Case 4: 987654304
1234567890 2345678901	Case 5: 3825876150
0 4294967295	

SOLUTION

```
using namespace std;
#include<bits/stdc++.h>
#define D(x) cout<< #x " = "<<(x)<<endl
#define cl(x) memset(x, -1,sizeof x);
template<class T> string to_str(T t){stringstream ss; ss<<t; return
ss.str();}
typedef long long ll;
ll a,b;
ll _dp[33][33][2][2];
ll dp(int index, int llevo, int tope, int puse, string &s){
    if(index == s.size()) return llevo;
    if(_dp[index][llevo][tope][puse] != -1) return
_dp[index][llevo][tope][puse];
    ll ans = 0;
    int m = ((tope)? (s[index] - '0') : 9);
    for(int i = 0 ; i <= m; ++i)
        ans += dp(index + 1 , (puse ? (llevo + (i==0)) : 0) , tope and (i ==
(s[index] - '0')), (i != 0) or puse, s);
    return _dp[index][llevo][tope][puse] = ans;
}
void read(){cin>>a>>b;a--;}
void solve(int Case){
    string A = to_str(a);
    string B = to_str(b);
    cl(_dp);
    ll from = (a == -1)? -1 : dp(0,0,1,0,A);
    cl(_dp);
    ll to = dp(0,0,1,0,B);
    cout<<"Case "<<Case<<": "<<(to - from)<<endl;
}
int main(){
    cin.sync_with_stdio(false);
    int t;cin>>t;
    for(int i = 0; i < t; ++i, read(),solve(i));
}
```


1145 - Dice (I)

Time Limit: 2 second(s)

Memory Limit: 32 MB

You have **N** dices; each of them has **K** faces numbered from **1** to **K**. Now you have arranged the **N** dices in a line. You can rotate/flip any dice if you want. How many ways you can set the top faces such that the summation of all the top faces equals **S**?

Now you are given **N, K, S**; you have to calculate the total number of ways.

Input

Input starts with an integer **T** (≤ 25), denoting the number of test cases.

Each case contains three integers: **N** ($1 \leq N \leq 1000$), **K** ($1 \leq K \leq 1000$) and **S** ($0 \leq S \leq 15000$).

Output

For each case print the case number and the result modulo **100000007**.

Sample Input	Output for Sample Input
5	Case 1: 1
1 6 3	Case 2: 7
2 9 8	Case 3: 57286574
500 6 1000	Case 4: 72413502
800 800 10000	Case 5: 9
2 100 10	

SOLUTION

```
using namespace std;
#include<bits/stdc++.h>
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
#define D(x) cout<< #x " = "<<(x)<<endl
const int MN = 15050;
const int mod = 100000007;
int N,S,K;
long long dp[2][MN];
void solve() {
    cin>>N>>K>>S;
    memset(dp, 0, sizeof dp);
    dp[1][0] = 1;
    int i = 0;
    for (int ii = N - 1; ii >= 0; --ii, i ^= 1) {
        dp[i][0] = dp[i][1] = 0;
        for (int j = 1; j <= S; ++j) {
            long long tmp = dp[i ^ 1][j - 1];
            if (j - K >= 0)
                tmp = (tmp - dp[i ^ 1][j - K - 1] + mod) % mod;
            dp[i][j] = tmp;
            if (j)
                dp[i][j] = (dp[i][j] + dp[i][j - 1]) % mod;
        }
    }
    long long ans = dp[i ^ 1][S];
    while (ans < 0) ans += mod;
    cout<<(ans)%mod<<endl;
}
int main() { __
    int t;cin>>t;
    for (int i = 0; i < t; ++i) {
        cout<<"Case "<<(i+1)<<": ";
        solve();
    }
    return 0;
}
```

1147 - Tug of War

Time Limit: 4 second(s)

Memory Limit: 32 MB

A tug of war is to be arranged at the local office picnic. For the tug of war, the picnickers must be divided into two teams. Each person must be on one team or the other; the number of people on the two teams must not differ by more than 1; the total weight of the people on each team should be as nearly equal as possible.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

The first line of each case is a blank line. The next line of input contains an integer n ($2 \leq n \leq 100$), the number of people at the picnic. n lines follow. The first line gives the weight of person 1; the second the weight of person 2; and so on. Each weight is an integer between 1 and 100000. The summation of all the weights of the people in a case will not exceed 100000.

Output

For each case, print the case number and the total number weights of the people in two teams. If the weights differ, print the smaller weight first.

Sample Input	Output for Sample Input
2	Case 1: 190 200
3	Case 2: 30 32
100	
90	
200	
4	
10	
15	
17	
20	

SOLUTION

```
#include <bits/stdc++.h>
#define ll long long
#define MAX 100100

using namespace std;

vector <ll > v;
map <pair<ll,ll>,ll>mp;
ll dp[503000];
ll mini;
ll as,bs;
#define BIT(n) (1LL<<(n))
int main()
{
    #ifndef ONLINE_JUDGE
    freopen("/home/sameer/Documents/sameer/input.sam","r",stdin);
    #endif

    ios_base::sync_with_stdio(false);
    ll i,j,k,n,m,t,cont,a,b;

    cin >> t;
    ll cases = t,ans;
    while(t--){
        v.clear();
        mp.clear();
        memset(dp,0,sizeof dp);
        cin >> n ;

        ll sum =0;
        for(i=0;i <n;i++) {
            cin >> k;
            v.push_back(k);
            sum+= k;
        }
        ll total = sum;
        sum /= 2;

        dp[0] =1;
        ll one =1;
        for(i =0;i < v.size();i++) {

            for(j = sum;j >= v[i];j--) {
```

```

        dp[j] = dp[j] | (dp[j-v[i]] << 1);
    }
}
a = n/2, b = n-a;
for( j= sum+1 ;j >=0;j--){
    //      cout <<j <<" "<< f[j] <<" " << BIT(a) << " " <<
BIT(b) << endl;
        if ((dp[j] & BIT(a)) || (dp[j] & BIT(b))) ;
            ans = j; break;
    }
    cout << "Case " << cases-t << ": " << min(ans,total-ans)
<< " " << max(total-ans,ans) << endl;
}
return 0;
}

```

1157 - LCS Revisited

Time Limit: 2 second(s)

Memory Limit: 32 MB

LCS means 'Longest Common Subsequence' that means two non-empty strings are given; the longest subsequence that are common. Subsequence means removing 0 or more characters from a string.

Now you are given two non-empty strings **s** and **t**, your task is to find the number of distinct LCS of **s** and **t**. Since the result can be very big, print the result modulo **1000007**.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case contains two lines, the first line is the string **s** and the second line is the string **t**. You may assume that the strings are non-empty and consist only of lowercase letters and the length of the each string is at most **1000**.

Output

For each case, print the case number and the number of distinct LCS of **s** and **t** modulo **1000007**.

Sample Input	Output for Sample Input
4 acbd acbd vnvn vn ab ba xyz abc	Case 1: 1 Case 2: 1 Case 3: 2 Case 4: 1

SOLUTION

```
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
using namespace std;
const int MN = 1010;
int lcs[MN][MN];
int dp[MN][MN];
int n_o[MN][30][3]; // next occ
const int mod = 1000007;
int add (int a, int b) {
    int ans = a + b;
    if (ans >= mod)
        ans -= mod;
    return ans;
}
int dfs(string &a, string &b, int i, int j) {
    if(lcs[i][j] == 0)
        return 1;
    if (dp[i][j] != -1) return dp[i][j];
    dp[i][j] = 0;
    for (int k = 0; k < 26; ++k) {
        int x = n_o[i][k][0], y = n_o[j][k][1];
        if (lcs[x][y] == lcs[i][j]) {
            dp[i][j] = add(dp[i][j], dfs(a, b, x + 1, y + 1));
        }
    }
    return dp[i][j];
}
void solve() {
    string a, b;
    cin >> a >> b;
    for (int k = 0; k < 26; ++k) {
        n_o[a.size()][k][0] = a.size();
        n_o[b.size()][k][1] = b.size();
    }
    for (int i = a.size() - 1; i >= 0; --i)
        for (int k = 0; k < 26; ++k)
            if (a[i] == (k + 'a')) n_o[i][k][0] = i;
            else n_o[i][k][0] = n_o[i + 1][k][0];
    for (int i = b.size() - 1; i >= 0; --i)
        for (int k = 0; k < 26; ++k)
            if (b[i] == (k + 'a')) n_o[i][k][1] = i;
            else n_o[i][k][1] = n_o[i + 1][k][1];
    for (int i = a.size(); i >= 0; --i) {
```

```

        for (int j = b.size(); j >= 0; --j) {
            if (i == a.size() || j == b.size()) {
                lcs[i][j] = 0;
                continue;
            }
            lcs[i][j] = max(lcs[i + 1][j], lcs[i][j + 1]);
            if (a[i] == b[j])
                lcs[i][j] = max(lcs[i][j], lcs[i + 1][j + 1] + 1);
        }
    }
    int ans;
    memset(dp, -1, sizeof dp);
    ans = dfs(a, b, 0, 0);
    /*for (int i = a.size(); i >= 0; --i) {
        for (int j = b.size(); j >= 0; --j) {
            if (lcs[i][j] == 0) {
                dp[i][j] = 1;
                continue;
            }
            dp[i][j] = 0;
            for (int k = 0; k < 26; ++k) {
                int x = n_o[i][k][0], y = n_o[j][k][1];
                if (lcs[x][y] == lcs[i][j]) {
                    dp[i][j] = add(dp[i][j], dp[x + 1][y + 1]);
                }
            }
        }
    }
    */
    printf("%d\n", ans);
}

int main () {
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int tc; cin >> tc;
    for (int i = 0; i < tc; ++i) {
        printf("Case %d: ", i + 1);
        solve();
    }
    return 0;
}

```


1158 - Anagram Division

Time Limit: 2 second(s)

Memory Limit: 32 MB

Given a string **s** and a positive integer **d** you have to determine how many permutations of **s** are divisible by **d**.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case contains a string **s** ($1 \leq \text{s}_{\text{length}} \leq 10$) and an integer **d** ($1 \leq d \leq 1001$). **s** will only contain decimal digits.

Output

For each case, print the case number and the number of permutations of **s** that are divisible by **d**.

Sample Input	Output for Sample Input
3 000 1 1234567890 1 123434 2	Case 1: 1 Case 2: 3628800 Case 3: 90

SOLUTION

```
#include <bits/stdc++.h>
#define ll long long
#define MAX 100100
using namespace std;
ll fact[11], countr[11];
char s[11];
ll dp[1 << 11][1005];
ll d;
ll glob_len_s;
ll permutations(int num, int curr)
{
    //cout << num << " " << curr << endl;
    int len = __builtin_popcount(num);
    if (num == (1 << glob_len_s) - 1) return curr ? 0 : 1;
    if (len > glob_len_s || num >= (1 << glob_len_s) )
        return 0;
    if (dp[num][curr] != -1)
        return dp[num][curr];
    dp[num][curr] = 0;
    for (int i = 0; i < glob_len_s; i++) {
        if ((num & (1 << i)) == 0) {
            int elem = s[i] - '0';
            dp[num][curr] += permutations(num | (1 << i),
(curr * 10 + elem) % d );
        }
    }
    return dp[num][curr];
}

int main()
{
#ifdef ONLINE_JUDGE
    freopen("/home/sameer/Documents/sameer/input.sam", "r", stdin);
#endif

    //ios_base::sync_with_stdio(false);
    ll i, j, k, n, m, t, cont, a, b;
    scanf("%lld", &t);
    ll cases = t ;
    fact[0] = 1;
    for (i = 1; i < 14; i++) {
        fact[i] = i * fact[i - 1];
    }
    while (t--) {
        memset(dp, -1, sizeof dp);
```

```

        memset(countr, 0, sizeof countr);
        scanf("%s %lld", s, &d);
        glob_len_s = strlen(s);
        for (i = 0; i < glob_len_s; i++) {
            countr[s[i] - '0']++;
        }
        ll ans = permutations(0, 0);
        ll divide = 1;
        for (i = 0; i <= 9; i++) {
            divide *= fact[countr[i]];
        }
        printf("Case %lld: %lld\n", cases - t, ans / divide);
    }
    return 0;
}

```

1159 - Batman

Time Limit: 2 second(s)

Memory Limit: 32 MB

Batman is in deep trouble. You know that superheroes are there to help you when you are in trouble. But in Gotham city there is no trouble. So, 'no trouble' is actually the trouble for our Batman.

So, Batman is trying to solve ACM problems because he wants to be a good programmer like you :). But alas! He is not that smart. But still he is trying. He found 3 strings of characters. Now he wants to find the maximum string which is contained in all the three strings as a sub sequence. He wants to find the maximum length, not the sequence.

Now, Batman claims that he is a better programmer than you. So, you are solving the same problem. Can you solve faster? You are guaranteed that Batman will need 3 hours to solve the problem. So, you have to be faster than him.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case will contain a blank line and three strings in three lines. None of the string lengths will be greater than **50** and less than **1**. And the string will contain alphanumeric characters only.

Output

For each case, print one line containing the case number and the length of the largest subsequence.

Sample Input	Output for Sample Input
3 abcdef cdef dcdef aaaa bbbb ccca aaaa aaaa aaa	Case 1: 4 Case 2: 0 Case 3: 3

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
int main()
{
    int t;
    int m;
    int n;
    int p;
    char a[100];
    char b[100];
    char c[100];
    int maxi;
    int ans[100][100][100];

    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%s", a);
        scanf("%s", b);
        scanf("%s", c);
        m = strlen(a);
        n = strlen(b);
        p = strlen(c);
        for (int i = m; i >= 0; i--) {
            for (int j = n; j >= 0; j--) {
                for (int k = p; k >= 0; k--) {

                    if(i == m or j == n or k == p) {
                        ans[i][j][k] = 0;
                        continue;
                    }
                    if(a[i] == b[j] and b[j] == c[k]) {
                        ans[i][j][k] = 1 +

ans[i+1][j+1][k+1];

                        continue;
                    }
                }
                maxi = -1;
                maxi = max(ans[i+1][j][k], maxi);
                maxi = max(ans[i][j+1][k], maxi);
                maxi = max(ans[i][j][k+1], maxi);
                maxi = max(ans[i+1][j+1][k], maxi);
                maxi = max(ans[i+1][j][k+1], maxi);
            }
        }
    }
}
```

```

                                maxi = max(ans[i][j+1][k+1], maxi);
                                ans[i][j][k] = maxi;
                            }
                        }
                    }
    printf("Case %d: %d\n", cs, ans[0][0][0]);
}
}
```

1169 - Monkeys on Twin Tower

Time Limit: 2 second(s)

Memory Limit: 32 MB

Being inspired by the ongoing popularity of animation films, the monkeys are trying to be smarter. They have realized that the only way to get smarter is to learn mathematics. Hence, they have started to do so. With the creative brains as they have, they are applying math in all aspects of life. Now, one of these mathematician monkeys are standing in front of a multi-storied twin tower. The twin tower is actually a couple of tall buildings standing parallel to each other. Each of the buildings has n floors. The ground floor is floor 0, the next one is floor 1 and so on. So, there are $2n$ floors in total in the twin tower. Each of these floors has a fruit inside it. The monkey knows in advance the amount of time required to eat the fruit in any floor. The monkey starts from the ground floor, climbs up toward the top of the buildings and has to eat exactly n fruits. From floor i , he has only two ways to go to floor $i + 1$. He can go the floor $i + 1$ of the same building that he is on in floor i . As he is a good jumper, it can be completed in no time. Also, he can go to floor $i + 1$ of the other building using a spiral stair connecting the two buildings. This will take a certain time. Please note that, the monkey can only move to floor $i + 1$ from floor i . He wants to figure out the minimum time required to eat n fruits. Can you verify how good his math is?

Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each test case consists of five lines. The first line has a single integer n ($1 \leq n \leq 1000$), the number of floors in each building. The 2nd line contains n integers separated by a single space. These integers denote the number of seconds required to eat the fruit in each floor for the first building. The time is given in ascending order of the floor i.e. the first integer is the number of seconds required to eat the fruit in ground floor of the first building while the last integer is the time required for the fruit in the topmost floor. The next line, containing n integers, describes the same values for the right building. Each of the above $2n$ integers has a value between 1 and 100. The line four has $n - 1$ space separated integers. These values denote the time required to jump from the left building to the right one. So, the first integer is the number of seconds to jump from ground floor left building to 1st floor right building. Finally, the fifth line contains $n - 1$ more integers giving the time required for jumping from the right building to the left one. The jumping times have values between 1 and 50.

Output

For each case, print the case number and the minimum number of seconds required to eat n fruits.

Sample Input	Output for Sample Input
1 4 5 6 8 9 7 9 3 10 5 2 3 2 4 3	Case 1: 26

SOLUTION

```
#include <iostream>
#include <cstdio>
#define ST 1002
#define TI 1002
using namespace std;
int main() {
    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);
    int i, j, n;
    int a1[ST];
    int a2[ST];
    int t1[TI];
    int t2[TI];
    int f1[ST], f2[ST];
    int T, t=1;
    scanf("%d", &T);
    while(T--) {
        scanf("%d", &n);
        for(i=1; i<=n; i++)
            scanf("%d", &a1[i]);
        for(i=1; i<=n; i++)
            scanf("%d", &a2[i]);
        for(i=1; i<n; i++)
            scanf("%d", &t1[i]);
        for(i=1; i<n; i++)
            scanf("%d", &t2[i]);
        f1[1]=a1[1];
        f2[1]=a2[1];
        for(j=2; j<=n; j++) {
            f1[j]=min(f1[j-1]+a1[j], f2[j-1]+a1[j]+t2[j-1]);
            f2[j]=min(f2[j-1]+a2[j], f1[j-1]+t1[j-1]+a2[j]);
        }
        printf("Case %d: %d\n", t, min(f1[n], f2[n]));
        t++;
    }
    return 0;
}
```

1170 - Counting Perfect BST

Time Limit: 2 second(s)

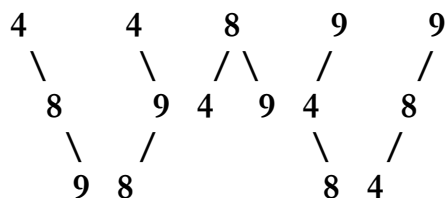
Memory Limit: 32 MB

BST is the acronym for Binary Search Tree. A BST is a tree data structure with the following properties.

- i) Each BST contains a root node and the root may have zero, one or two children. Each of the children themselves forms the root of another BST. The two children are classically referred to as left child and right child.
- ii) The left subtree, whose root is the left children of a root, contains all elements with key values less than or equal to that of the root.
- iii) The right subtree, whose root is the right children of a root, contains all elements with key values greater than that of the root.

An integer m is said to be a perfect power if there exists integer $x > 1$ and $y > 1$ such that $m = x^y$. First few perfect powers are $\{4, 8, 9, 16, 25, 27, 32, 36, 49, 64, 81, 100, 121, 125, 128, 144, \dots\}$. Now given two integer a and b we want to construct BST using all perfect powers between a and b , where each perfect power will form the key value of a node.

Now, we can construct several BSTs out of the perfect powers. For example, given $a = 1$ and $b = 10$, perfect powers between a and b are 4, 8, 9. Using these we can form the following five BSTs.



In this problem, given a and b , you will have to determine the total number of BSTs that can be formed using perfect powers between a and b .

Input

Input starts with an integer T (≤ 20000), denoting the number of test cases.

Each case of input contains two integers: a and b ($1 \leq a \leq b \leq 10^{10}$, $b - a \leq 10^6$) as defined in the problem statement.

Output

For each case, print the case number and the total number of distinct BSTs that can be formed by the perfect powers between a and b . Output the result modulo 100000007.

Sample Input	Output for Sample Input
4	Case 1: 1
1 4	Case 2: 2
5 10	Case 3: 5
1 10	Case 4: 0
1 3	

SOLUTION

```
using namespace std;
#include<bits/stdc++.h>
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
#define D(x) cout<< #x " = "<<(x)<<endl
#define MP 1000010
const int mod = 1000000007;
long long p[203800];
int c;
void gen_powers(){
    c = 0;
    for(long long i = 2; i <= 100001; ++i ){
        long long num = i*i;
        while(num <= 100000000001L){
            p[c++] = num;
            num*=i;
        }
    }
    sort(p,p+c);
    c = unique(p,p+c) - p;
    p[c++] = 10001000000;
}
long long cat[MP];
long long fac[2*MP];
long long mod_pow(int n, int m = (mod - 2)){
    long long ans = 1;
    for(long long base = n ; m ; m>>=1){
        if(m&1) ans = (ans * base)%mod;
        base = (base * base)%mod;
    }
    return ans;
}
void fill_cat(){
    fac[0] = fac[1] = 1;
    for(int i = 2; i < 2*MP; ++i)
        fac[i] = (i*fac[i-1])%mod;
    cat[0] = cat[1] = 1;
    for(int i = 2; i < MP; ++i){
        cat[i] = fac[2*i];
        long long tmp = (fac[i+1]*fac[i]) % mod;
        tmp = mod_pow(tmp);
        cat[i] = (cat[i]*tmp)%mod;
    }
}
```

```

int solve(){
    long long a,b;
    cin>>a>>b;
    int f = lower_bound(p, p+c, a) - p;
    int t = lower_bound(p, p+c, b) - p;
    if(p[t] > b) t--;
    int n = t - f + 1;
    if(!n) return 0;
    return cat[n];
}

int main(){ ____
    gen_powers();
    fill_cat();
    /*
    for(int i = 0; i < 10; ++i){
        D(fac[i]);
        D(cat[i]);
    }
    */
    int tc;cin>>tc;
    for(int i = 0; i< tc; ++i)
        printf("Case %d: %d\n",i+1,solve());
    return 0;
}

```

1173 - The Vindictive Coach

Time Limit: 2 second(s)

Memory Limit: 32 MB

The coach of a football team, after suffering for years the adverse comments of the media about his tactics, decides to take his revenge by presenting his players in a line-up in such a way that the TV cameras would be compelled to zigzag in a ridiculous bobbing motion, by alternating taller and shorter players. However, the team captain objects that he must be the first of the line by protocolary reasons, and that he wants to be seen in the best possible light: that is, he should not have a taller colleague next to him unless there is no alternative (everyone else is taller than him). Even in this case, the height difference should be as small as possible, while maintaining the zigzag arrangement of the line.

With this condition the coach addresses an expert in computation (i.e. you) to help him find the number of different alignments he may make, knowing that all players have a different height. They are always numbered by stature starting by 1 as the shortest one. Of course the number of players may be arbitrary, provided it does not exceed 50.

Input

Input starts with an integer T (≤ 125), denoting the number of test cases.

Each case contains two positive integers N ($1 \leq N \leq 50$) and m ($1 \leq m \leq N$) separated by a blank space. N represents the number of players in the line-up and m is the captain's number, who as told is always the first of the line.

Output

For every case, print the case number and the number of possible alignments under the above conditions. Output the result modulo 2^{64} .

Sample Input	Output for Sample Input
4	Case 1: 1
3 1	Case 2: 1
3 3	Case 3: 1
4 1	Case 4: 16
7 2	

SOLUTION

```
const int NX = 55 ;
int cs , vis[NX][NX][2] , n , m ;
bool taken[NX] ;
ull dp[NX][NX][2];
ull DP( int pos , int lst , int isSmall )
{
    if( pos == n ) return 1ll ;
    int &v = vis[pos][lst][isSmall];
    ull &ret = dp[pos][lst][isSmall];
    if( v == cs ) return ret ;
    v = cs ;
    ret = 0 ;
    int i ;
    if( isSmall )
    {
        for ( i = 1 ; i < lst ; i++ )
        {
            if( taken[i] ) continue ;
            taken[i] = 1 ;
            ret += DP( pos + 1 , i , 0 );
            taken[i] = 0 ;
        }
    }
    else
    {
        for( i = lst + 1 ; i <= n ; i++ )
        {
            if( taken[i] ) continue ;
            taken[i] = 1 ;
            ret += DP( pos + 1 , i , 1 );
            taken[i] = 0 ;
        }
    }
    return ret ;
}

int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // cout << fixed << setprecision(10) ;
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
```

```

n = II , m = II ;
if( n < 3 )
{
    printf("Case %d: 1\n",cs);
}
else
{
    ull ans ;
    ms( taken , 0 );
    if( m == 1 )
    {
        taken[m] = 1 ;
        taken[3] = 1 ;
        ans = DP( 2 , 3 , 1 );
    }
    else
    {
        taken[m] = 1 ;
        ans = DP( 1 , m , 1 );
    }
    printf("Case %d: %llu\n",cs,ans);
}
}
return 0;
}

```


1180 - Software Company

Time Limit: 2 second(s)

Memory Limit: 32 MB

A software developing company has been assigned two programming projects. As both projects are within the same contract, both must be handed in at the same time. It does not help if one is finished earlier.

This company has n employees to do the jobs. To manage the two projects more easily, each is divided into m independent subprojects. Only one employee can work on a single subproject at one time, but it is possible for two employees to work on different subprojects of the same project simultaneously. Our goal is to finish the projects as soon as possible.

Input

Input starts with an integer T (≤ 12), denoting the number of test cases.

Each case starts with two integers n ($1 \leq n \leq 100$), and m ($1 \leq m \leq 100$). Each of the next n lines contains two integers which specify how much time in seconds it will take for the specified employee to complete one subproject of each project. So if the line contains x and y , it means that it takes the employee x seconds to complete a subproject from the first project, and y seconds to complete a subproject from the second project.

Output

For each case, print the case number and the minimum amount of time in seconds after which both projects can be completed. The input will be such that answer will be within 50000.

Sample Input	Output for Sample Input
1 3 20 1 1 2 4 1 6	Case 1: 18

SOLUTION

```
bool dp[104][103][103] ;
int done[104][103][103] ,cc=1;
int n,m,mid,l,r;
vector<pair<int,int > > kp(103) ;
bool re(int p,int c1,int c2){
    if(p==n)
        return (c1==c2 and c1==0) ;
    bool &res = dp[p][c1][c2];
    if(done[p][c1][c2]==cc)
        return res;
    done[p][c1][c2]= cc;
    res = false;
    for(int i=0;i<=c1 and !res and i*kp[p].F<=mid;i++){
        int S = (mid - i*kp[p].F)/kp[p].S;
        res = re(p+1,c1-i,c2 - min(S,c2) );
    }
    return res;
}
int main()
{
    //      std::ios_base::sync_with_stdio(false);
    #ifndef kimbbakar
        freopen ( "in.txt", "r", stdin );
    //      freopen ( "out.txt", "w", stdout );
    #endif

    int t,tcase=1;
    in(t);
    while(t--){
        in(n),in(m);
        for(int i=0;i<n;i++){
            in(kp[i].F);
            in(kp[i].S);
        }
        l = 0,r = 50000;
        int res = r;
        bool tmp;
        mid = 17;
        while(l<r){
            mid = (l+r)>>1;
            tmp = re(0,m,m);
            if(tmp)
                res = min(res,mid);
        }
    }
}
```

```

        cc++;
        if(l==mid){
            mid = r;
            tmp = re(0,m,m);
            if(tmp)
                res = min(res,mid);
            cc++;
            break;
        }
        if(tmp)
            r = mid;
        else l = mid;
    }
    pf("Case %d: %d\n",tcase++,res);
}
return 0;
}

```

1191 - Bar Codes

Time Limit: 0.5 second(s)

Memory Limit: 32 MB

A bar-code symbol consists of alternating dark and light bars, starting with a dark bar on the left. Each bar is a number of units wide. Figure 1 shows a bar-code symbol consisting of 4 bars that extend over $1+2+3+1=7$ units.

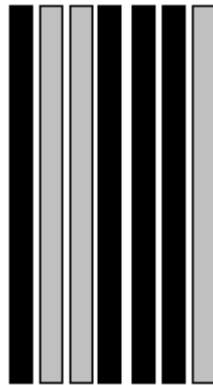


Figure 1: Bar-code over 7 units with 4 bars

In general, the bar code $\mathbf{BC(n, k, m)}$ is the set of all symbols with \mathbf{k} bars that together extend over exactly \mathbf{n} units, each bar being at most \mathbf{m} units wide. For instance, the symbol in Figure 1 belongs to $\mathbf{BC(7,4,3)}$ but not to $\mathbf{BC(7,4,2)}$. Figure 2 shows all 16 symbols in $\mathbf{BC(7,4,3)}$. Each '1' represents a dark unit, each '0' represents a light unit.

0: 1000100 | 4: 1001110 | 8: 1100100 | 12: 1101110
1: 1000110 | 5: 1011000 | 9: 1100110 | 13: 1110010
2: 1001000 | 6: 1011100 | 10: 1101000 | 14: 1110100
3: 1001100 | 7: 1100010 | 11: 1101100 | 15: 1110110

Figure 2: All symbols of $\mathbf{BC(7,4,3)}$

Input

Input starts with an integer \mathbf{T} (≤ 20000), denoting the number of test cases.

Each case contains three integers: $\mathbf{n, k, m}$ ($1 \leq \mathbf{k, m} \leq \mathbf{n} \leq 50$).

Output

For each case, print the case number and $\mathbf{BC(n, k, m)}$.

Sample Input	Output for Sample Input
2	Case 1: 16
7 4 3	Case 2: 4
7 4 2	

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
const int MN = 51;
long long dp[MN][MN][MN];
long long go(int i, int k, int m) {
    if (k < 0 || i < 0)
        return 0;
    if (i == 0)
        return k == 0;
    if (dp[i][k][m] != -1)
        return dp[i][k][m];
    long long ans = 0;
    for (int j = 1; j <= m; ++j) {
        ans += go(i - j, k - 1, m);
    }
    return dp[i][k][m] = ans;
}
void solve() {
    int n, k, m;
    cin >> n >> k >> m;
    long long ans = go(n, k, m);
    printf("%lld\n", ans);
}
int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int tc; cin >> tc;
    memset(dp, -1, sizeof dp);
    for (int i = 0; i < tc; ++i) {
        printf("Case %d: ", i + 1);
        solve();
    }
    return 0;
}
```

1193 - Dice (II)

Time Limit: 3 second(s)

Memory Limit: 32 MB

You have N dices; each of them has K faces numbered from 1 to K . Now you can arrange the N dices in a line. If the summation of the top faces of the dices is S , you calculate the score as the multiplication of all the top faces.

Now you are given N, K, S ; you have to calculate the summation of all the scores.

Input

Input starts with an integer T (≤ 25), denoting the number of test cases.

Each case contains three integers: N ($1 \leq N \leq 1000$) K ($1 \leq K \leq 1000$) S ($0 \leq S \leq 15000$).

Output

For each case print the case number and the result modulo 100000007 .

Sample Input	Output for Sample Input
5	Case 1: 3
1 6 3	Case 2: 84
2 9 8	Case 3: 74335590
500 6 1000	Case 4: 33274428
800 800 10000	Case 5: 165
2 100 10	

SOLUTION

```
#include <bits/stdc++.h>
#define ll long long
#define mod 100000007
using namespace std;
ll sum[15110];
ll dp[2][15110];
int main()
{
#ifdef ONLINE_JUDGE
    freopen("/home/sameer/Documents/sameer/input.sam", "r", stdin);
#endif
    ios_base::sync_with_stdio(false);
    ll i, j, k, n, m, t, cont, a, b, s;
    cin >> t;
    ll cases = t;
    while (t--) {
        cin >> n >> k >> s;
        memset(sum, 0, sizeof sum);
        memset(dp, 0, sizeof dp);
        for (i = 1; i <= k; i++)
            dp[1][i] = i;
        for (i = 2; i <= n; i++) {
            ll temp = i & 1;
            memset(sum, 0, sizeof sum);
            for (j = 1; j <= s; j++) {
                sum[j] = (sum[j - 1] + dp[temp ^ 1][j]) %
mod;

                dp[temp][j] = (dp[temp][j - 1] + sum[j -
1]) % mod;

                if (j - 1 - k >= 0) {
                    dp[temp][j] = (((dp[temp][j] -
sum[j - 1 - k]) % mod + mod) % mod);
                    dp[temp][j] = (((dp[temp][j] -
dp[temp ^ 1][j - 1 - k] * k) % mod + mod) % mod);
                }
            }
        }
        cout << "Case " << cases - t << ": " << dp[n & 1][s] <<
endl;
    }
    return 0;
}
```


1194 - Colored T-Shirts

Time Limit: 2 second(s)

Memory Limit: 32 MB

N students are standing in a line. They are wearing colored t-shirts. For simplicity, let's assume that the color of each person is an integer between **1** and **16**. Like color 1 represents red, color 2 represents blue, and so on.

Now the students with same colored t-shirts want to join together. So, they are thinking what to do. That's why you are here to the rescue! You have to solve this problem. You have to say the minimum number of swaps necessary to reformat the line such that the people with same colored t-shirt come together. A swap means changing position between two adjacent people in the line.

Input

Input starts with an integer T (≤ 15), denoting the number of test cases.

Each case starts with two integers N ($1 \leq N \leq 10^5$) and m ($1 \leq m \leq 16$), where m denotes the number of total colors. So, the students in the line will have a t-shirt color between **1** and m . The next line contains N space separated integers. Each of these integers will lie between **1** and m , denoting the color of the i^{th} person in the line.

Output

For each case, print the case number and the minimum number of swaps necessary to arrange them according to the description.

Sample Input	Output for Sample Input
3	Case 1: 1
4 2	Case 2: 6
1 2 1 2	Case 3: 5
6 4	
2 1 4 3 1 2	
8 6	
1 3 2 5 5 4 5 2	

Note

Dataset is huge. Use faster i/o methods.

SOLUTION

```
const int NX = 1e5 + 10 ;
Long dp[ NX ] , pre[ NX ][ 18 ] , freq[ NX ];
int n , m ;
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II , m = II ;
        ms( pre , 0 );
        ms( freq , 0 );
        for ( int i = 0 ; i < n ; i++ )
        {
            int x = II ;
            x-- ;
            for ( int j = 0 ; j < m ; j++ )
            {
                pre[x][j] += freq[j];
            }
            freq[x]++;
        }
        int lim = ( 1 << m ) ;
        for( int i = 0 ; i < lim ; i++ )
        {
            dp[i] = 1e17 ;
        }
        dp[0] = 0 ;
        for ( int i = 0 ; i < lim ; i++ )
        {
            for ( int j = 0 ; j < m ; j++ )
            {
                if( i & ( 1 << j ) ) continue ;
                Long add = 0 ;
                for ( int k = 0 ; k < m ; k++ )
                {
                    if( i & ( 1 << k ) ) continue ;
                    if( j == k ) continue ;
                    add += pre[j][k];
                }
                dp[ i | ( 1 << j ) ] = min( dp[ i | ( 1 << j ) ] , dp[i] + add
            );
        }
    }
}
```

```
        }  
    }  
    printf("Case %d: %lld\n",cs,dp[lim-1]);  
}  
return 0 ;  
}
```

1200 - Thief

Time Limit: 3 second(s)

Memory Limit: 32 MB

A thief has entered into a super shop at midnight. Poor thief came here because his wife has sent him to buy some necessary households. Instead of buying the items, he decided to steal them.

He has a bag with him which can carry up to W kg. In the list (given by his wife) there are four fields 1) item name, 2) the price of the item, 3) how many of this item is required and 4) the weight of this item. The items are solid items, so he can't take any item after dividing into pieces.

Now the thief wants to take items in his bag such that it fulfills his wife's list, and the total weight of the items is not greater than W . And he can't take any item other than the items mentioned in the list, otherwise his wife would found the item and he may get caught. But he can take more items than required. And he wants to sell these extra items and earn some money. Assume that he can take any item (is in the list) any number of times unless they overflow his bag.

Now you are given the necessary information of the items and his bag, you have to find the maximum profit the thief can make.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing two integers n ($1 \leq n \leq 100$) and W ($1 \leq W \leq 10000$), where n denotes the number of items. Each of the next n lines contains three integers p_i c_i w_i ($1 \leq p_i, c_i, w_i \leq 100$), meaning that the price of the i^{th} item is p_i , c_i of it must be taken, and the weight is w_i .

Output

For each case, print the case number and the maximum profit he can get. If it's impossible to fulfill his wife's requirements, print 'Impossible'.

Sample Input	Output for Sample Input
2 3 20 10 1 10 5 1 5 1 1 1 1 10 10 1 11	Case 1: 4 Case 2: Impossible

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
int n;
int p[110];
int w[110];
int dp[10005][105];
int W;
int explore(int sum, int i)
{
    int x;
    int y;
    int z;

    x = y = z = 0;
    if(i == n) {
        return 0;
    }
    if(dp[sum][i] != -1) {
        return dp[sum][i];
    }

    if(sum + w[i] <= W)
        x = p[i] + explore( sum + w[i], i+1);

    if(sum + w[i] <= W)
        y = p[i] + explore(sum + w[i], i);
    z = explore(sum, i+1);
    return dp[sum][i] = max(x, max(y, z));
}
int main()
{
    int t;
    int x;
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        scanf("%d", &W);
        for (int i = 0; i < n; i++) {
            scanf("%d", &p[i]);
            scanf("%d", &x);
            scanf("%d", &w[i]);
        }
    }
}
```

```
        W = W - w[i] * x;
    }
    if(W < 0) {
        printf("Case %d: Impossible\n", cs);
        continue;
    }
    memset(dp, -1, sizeof(dp));
    printf("Case %d: %d\n", cs, explore(0, 0));
}
}
```

1201 - A Perfect Murder

Time Limit: 2 second(s)

Memory Limit: 32 MB

"Yes, I am the murderer. No doubt" I had to confess it in front of all. But wait, why I am confessing? Nobody wants to go to jail, neither do I. As you have suspected there is something fishy. So, let me explain a bit.

The murder was happened in 19th June, at 11:30 pm this year (2009) according to the medical report. So, I was asking the judge "Can you find the time 19th June 11:30 pm in Bangladesh?" The judge informed other reporters to find the time. But alas! There was no time - "2009, 19th June, 11:30 pm". So, the judge got a bit confused about my confession. So, I began to tell them, "The time the murder was happened, is not a valid time according to you. So, how can you claim that I am the murderer?"

And what happened next, you all know. I am in the streets again with a clean sheet.

But now I have planned to kill again. I have a list of N mosquitoes which are to be killed. But there is a small problem. If I kill a mosquito, all of his friends will be informed, so they will be prepared for my attack, thus they will be impossible to kill. But there is a surprising fact. That is if I denote them as a node and their friendship relations as edges, the graph becomes acyclic.

Now I am planning when and how to kill them (how to get rid of the law!) and you have to write a program that will help me to find the maximum number of mosquito I can kill. Don't worry too much, if anything goes wrong I will not mention your name, trust me!

Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each case starts with a blank line and two integers N ($1 \leq N \leq 1000$) denoting the number of mosquito I want to kill and M denoting the number of friendship configurations. Each of the next M lines contains two integers a and b denoting that a^{th} and b^{th} mosquitoes are friends. You can assume that ($1 \leq a, b \leq N, a \neq b$) and each friendship relation is given only once. As I have already mentioned, you will not find any cycle in the relations.

Output

For each case, print the case number and the maximum number of mosquitoes I can kill considering the conditions described above.

Sample Input	Output for Sample Input
3	Case 1: 3
4 3	Case 2: 2
1 2	Case 3: 3

1 3	
1 4	
3 2	
1 2	
2 3	
5 4	
1 2	
1 3	
2 4	
2 5	

SOLUTION

```
int
N,adj[1005][1005],deg[1005],dp[1005][2],M;
bool vis[1005][2],col[1005];
int DP(int u,int isGurd,int perent)
{
    // if(deg[u]==0) return 1;
    if(vis[u][isGurd]) return
dp[u][isGurd];
    vis[u][isGurd]=1;
    col[u]=1;
    int i,sum=0;
    for(i=0;i<deg[u];i++)
    {
        int v=adj[u][i];
        if(v!=perent)
        {
            if(!isGurd)
sum+=DP(v,1,u) ; // bosatei hobe
            else
sum+=min(DP(v,1,u),DP(v,0,u));
        }
    }
    return dp[u][isGurd]=sum+isGurd;
}
int main()
{
    //freopen("in.txt", "r", stdin);
    int cs,t;
    scanf("%d",&t);
    for(cs=1;cs<=t;cs++)
    {
        int i,j,u,v;
        scanf("%d %d",&N,&M);
        for(i=1;i<=N;i++) { deg[i]=0;
vis[i][0]=0; vis[i][1]=0; col[i]=0;}
        for(i=1;i<=M;i++)
        {
            scanf("%d %d",&u, &v);
            adj[u][deg[u]++]=v;
            adj[v][deg[v]++]=u;
        }
        int ans=0;
        for(i=1;i<=N;i++)
```

```
        {           if(!col[i])
                    ans+=min(DP(i,0,-
1),DP(i,1,-1));
        }
        printf("Case %d: %d\n",cs,N-
ans);
    }
}
```

1205 - Palindromic Numbers

Time Limit: 2 second(s)

Memory Limit: 32 MB

A palindromic number or numeral palindrome is a 'symmetrical' number like 16461 that remains the same when its digits are reversed. In this problem you will be given two integers i j , you have to find the number of palindromic numbers between i and j (inclusive).

Input

Input starts with an integer T (≤ 200), denoting the number of test cases.

Each case starts with a line containing two integers i j ($0 \leq i, j \leq 10^{17}$).

Output

For each case, print the case number and the total number of palindromic numbers between i and j (inclusive).

Sample Input	Output for Sample Input
4	Case 1: 9
1 10	Case 2: 18
100 1	Case 3: 108
1 1000	Case 4: 198
1 10000	

SOLUTION

```
typedef long long int __i64;
__i64 p , q , dp[20][2][2] , vis[20][2][2] , test;
int a[20], b[20] , maping , len;
__i64 DP(int pos, int gotSmall, int needSmall)
{
    if(pos==(len+1)/2)
    {
        //printf("len:%d needSmall:%d\n", len, needSmall);
        if(needSmall) return gotSmall; // jekhon match ta calculation er somoy
        pore boro kisu bosano hoyeche
        return 1;
    }
    __i64 &ret = dp[pos][gotSmall][needSmall];
    if(ret!=-1) return ret;
    // vis[pos][gotSmall][needSmall] = test;
    if(gotSmall)
    {
        ret = 10*DP(pos+1, 1, 0);
        return ret;
    }
    ret = 0;
    int i , s = ( pos==0 && len>1 ) ? 1 : 0;
    for( i = s; i < a[pos] ; i++ ) ret += DP(pos+1, 1, 0);
        if( i < a[len-1-pos] ) ret += DP(pos+1, 0, 0);
        else if( i == a[len-1-pos] ) ret += DP(pos+1, 0, needSmall);
        else ret += DP(pos+1, 0, 1);
    return ret;
}
__i64 solve(__i64 p)
{
    if(p==0) return 1;
    if(p<10) return p+1;
    int idx=0 , i;
    while(p)
    {
        b[idx++]=p%10;
        p/=10;
    }
    reverse(b, b+idx);
    __i64 ans = 0;
    for( i = 0 ; i < idx - 1 ; i++ )
    {
        a[i] = 9;
        len = i + 1;
        memo(dp, -1);
    }
}
```

```

        ans += DP(0,0,0);
        test++;
        //printf("len:: %d ans::%lld\n",len,ans);
    }
    for(i=0;i<idx;i++)
    {
        a[i] = b[i];
    }
    memo(dp,-1);
    len = idx;
    ans += DP(0,0,0);
    test++;
    return ans;
}

int main()
{
    // freopen("input.txt","r",stdin);
    int cs , t ;
    test = 1;
    scanf("%d",&t);
    for ( cs = 1 ,maping=1 ; cs <= t ; cs++ )
    {
        scanf("%lld %lld",&p,&q);
        if(p>q) swap(p,q);
        printf("Case %d: %lld\n",cs,solve(q) - solve(p-1));
    }
    return 0;
}

```

1217 - Neighbor House (II)

Time Limit: 2 second(s)

Memory Limit: 32 MB

A soap company wants to advertise their product in a local area. In this area, there are n houses and the houses are placed in circular fashion, such that house 1 has two neighbors: house 2 and n . House 5 has two neighbors: house 4 and 6. House n has two neighbors, house $n-1$ and 1.

Now the soap company has an estimation of the number of soaps they can sell on each house. But for their advertising policy, if they sell soaps to a house, they can't sell soaps to its two neighboring houses. Now your task is to find the maximum number of estimated soaps they can sell in that area.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing an integer n ($2 \leq n \leq 1000$). The next line contains n space separated integers, where the i^{th} integer denotes the estimated number of soaps that can be sold to the i^{th} house. Each of these integers will lie in the range $[1, 1000]$.

Output

For each case, print the case number and the maximum number of estimated soaps that can be sold in that area.

Sample Input	Output for Sample Input
3 2 10 100 3 10 2 11 4 8 9 2 8	Case 1: 100 Case 2: 11 Case 3: 17

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#define N 1000
using namespace std;
int a[N + 5];
long long dp[N+5][2];
int n;
long long explore(int i, int one)
{
    long long x;
    long long y;
    if(i >= n) {
        return 0;
    }
    if(i == n-1 and one == 1) {
        return 0;
    }

    if(dp[i][one] != -1)
        return dp[i][one];
    x = explore(i+1, one);
    if(i == 0) {
        one = 1;
    }
    y = explore(i+2, one) + a[i];
    return dp[i][one] = max(x, y);
}

int main()
{
    int t;

    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        memset(dp, -1, sizeof dp);
        for (int i = 0; i < n; i++) {
            scanf("%d", a + i);
        }

        printf("Case %d: %lld\n", cs, explore(0, 0));
    }
}
```


1223 - Testing Mailboxes

Time Limit: 2 second(s)

Memory Limit: 32 MB

When monkeys are given some fire-crackers, they have only thing in the mind - to blow things up. Small boxes were easy to blow up, and thus mailboxes became a popular target. Now, a small mailbox manufacturer is interested in how many fire-crackers his new mailbox prototype can withstand without exploding and has hired you to help him. He will provide you with **k** identical mailbox prototypes each fitting up to **m** fire-crackers. However, he is not sure of how many fire-crackers he needs to provide you with in order for you to be able to solve his problem, so he asks your help.

The constraints are:

1. If you blow up a mailbox, you can't use the mailbox again, so if you have only **k = 1** mailboxes, you would have to start testing with **1** fire-cracker, then **2** fire-crackers, and so on until it finally exploded. In the worst case, that is if it does not blow up even when filled with **m** fire-crackers, you would need $1 + 2 + 3 + \dots + m = m * (m + 1) / 2$ fire-crackers.
2. If a mailbox can withstand **x** fire-crackers, it can also withstand **x-1** fire-crackers.
3. Upon an explosion, a mailbox is either totally destroyed (blown up) or unharmed, which means that it can be reused in another test explosion.

Now the manufacturer wants you to find the maximum number of fire-crackers that his mailboxes can withstand. Before doing that you have to buy some fire-crackers to test that. So, you need to find the minimum number of fire-crackers you need to buy to test the mailboxes.

Input

Input starts with an integer **T** (≤ 10000), denoting the number of test cases.

Each case starts with a line containing two integers: **k** ($1 \leq k \leq 100$) and **m** ($1 \leq m \leq 100$).

Output

For each case, print the case number and the minimum number of fire-crackers you have to buy.

Sample Input	Output for Sample Input
4	Case 1: 55
1 10	Case 2: 382
3 73	Case 3: 495
5 100	Case 4: 5050
1 100	

SOLUTION

```
int dp[104][104][104] ;
bool ok[104][104][104] ;
int re(int x,int l,int r)
{
    if(l>r)
        return 0;
    if(x==0) return inf;
    int &res=dp[x][l][r] ;
    if(ok[x][l][r] ) return res;
    ok[x][l][r] =true;
    res=inf;
    pair<int,int>tmp1;
    pair<int,int>tmp2;
    pair<int,int>tmp3;
    for(int i=l;i<=r;i++)
    {
        res=min(res,i+max(re(x-1,l,i-1),re(x,i+1,r) ) ) ;
    }
    return res;
}
int main()
{
    //      std::ios_base::sync_with_stdio(false);
    #ifndef kimbbakar
        freopen ( "in.txt", "r", stdin );
    //      freopen ( "out.txt", "w", stdout );
    #endif
    int t,tcase=1;
    int m,k;
    in(t);
    while(t--)
    {
        in(m),in(k);
        pf("Case %d: %d\n",tcase++,re(m,1,k) );
    }
    return 0;
}
```

1226 - One Unit Machine

Time Limit: 2 second(s)

Memory Limit: 32 MB

OUM is a one unit machine which processes jobs. Since it can't handle heavyweight jobs; jobs needs to be partitioned into units. Initially, all the job information and unit partitions are given as input. Then the machine allocates necessary time slots. And in each time slot it asks the user for the name of the job to be processed. After getting the name; the machine determines the next unprocessed unit of that job and processes that unit in that slot. If there is no such unit, the machine crashes. A job is said to be complete if all the units of that job are complete.

For example, let J_1 and J_2 be two jobs each having 2 units. So, OUM will create 4 time slots. Now the user can give $J_1 J_2 J_2 J_1$ as input. That means it completes the 1st unit of J_1 in time slot 1 and then completes the 1st unit of J_2 in time slot 2. After that it completes the 2nd unit of J_2 and 2nd unit of J_1 in time slots 3 and 4 respectively. But if the user gives $J_1 J_1 J_2 J_1$ as input, the machine crashes in time slot 4 since it tries to process 3rd unit of J_1 which is not available.

Now, Sam is the owner of a software firm named ACM and he has n jobs to complete using OUM. He wants to complete Job_i before Job_{i+1} where $1 \leq i < n$. Now he wants to know the total number of ways he can complete these jobs without crashing the OUM. He assigned you for this task. Two ways are different if at t^{th} slot one processed a unit of Job_i and another processed a unit of Job_j where $i \neq j$. For the example above, there are three ways:

$J_1 J_1 J_2 J_2$
 $J_1 J_2 J_1 J_2$
 $J_2 J_1 J_1 J_2$

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with an integer n ($1 \leq n \leq 1000$). The next line contains n space separated positive integers $k_1, k_2, k_3 \dots k_n$. Where, k_i denotes the number of units for the i^{th} job. You can assume that total number of units for all the jobs in any case is not greater than 10^6 .

Output

For each case, print the case number and the result modulo 1000,000,007.

Sample Input	Output for Sample Input
2	Case 1: 3
2	Case 2: 45
2 2	
3	

2 2 3	
-------	--

SOLUTION

```
const Long mod = 1000000007 ;
const int NX = 1e6 + 10 ;
Long fact[NX];
Long BigMod( Long a , Long b )
{
    Long x = 1 ;
    while(b)
    {
        if( b & 1 ) x = ( x * a )%mod;
        b/=2 ;
        a = ( a * a)%mod;
    }
    return x ;
}
Long modInv( Long a)
{
    return BigMod(a , mod -2 );
}
void pre()
{
    fact[0] = 1 ;
    fact[1] = 1 ;
    int i ;
    for ( i = 2 ; i <= NX ; i++ )
    {
        fact[i] = ( fact[i-1] * i )%mod ;
    }
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    pre();
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        int n = II ;
        Long sum = 0 , ans = 1 ;
        rep( i , n )
        {
            Long x = LL ;
            sum += x ;
            ans = ( ans * fact[sum-1])%mod ;
        }
    }
}
```

```
        ans = ( ans * modInv((fact[sum-x] * fact[x-1] )%mod))%mod ;
    }
    printf("Case %d: %lld\n",cs,ans%mod);
}
return 0;
}
```


1228 - e-Friends

Time Limit: 2 second(s)

Memory Limit: 32 MB

Social networking web sites are very popular these days. I am not mentioning any names because you know better than me. People do have a lot of e-friends now, "you may not know his real identity but he is your friend". Sometimes, there can be rivalries between people, and thus people gets blocked or deleted. And these rivalries don't have to be necessarily commutative. That means x may think y as his enemy, but y may or may not think so.

However, what if you get the opportunity to see all your e-friends live? Since you like social networks very much, you planned to arrange a get-together with your e-friends. So, you invited n of your e-friends in your home. For simplicity, you numbered them from 1 to n . They were your friends, but rivalries existed amongst many of them. You asked them to form a queue such that they can get food from the table. And you will serve foods one by one. But your friends demanded that, no enemy should be the next person in the queue. That means if x thinks that y is his enemy and x 's position in the queue is i , then y 's position shouldn't be $i-1$.

So, it became very complex. That's why you decided to add a dissatisfaction index k if one's demand is not fulfilled. And the total dissatisfaction index is the summation of all the dissatisfaction indexes. Now you know the rivalries and the maximum total dissatisfaction index you may allow, you want to find the total number of possible arrangements.

Input

Input starts with an integer T (≤ 30), denoting the number of test cases.

Each case starts with three integers n ($1 \leq n \leq 12$), k ($0 \leq k \leq 10^6$) and q ($1 \leq q \leq 1000$) which denotes the number of queries. Then there will be n lines. The i^{th} line contains an integer t_i ($0 \leq t_i < n$) and ids of t_i distinct friends. It means i^{th} person thinks the t_i listed persons as his enemy. Each of the next q lines contains an integer r ($0 \leq r \leq 10^8$) denoting the total allowable dissatisfaction index.

Output

For each case, print the case number first. Then for each query, print the total number of arrangements possible not exceeding the given total dissatisfaction index.

Sample Input	Output for Sample Input
1 2 10 2 1 2 0 10 5	Case 1 : 2 1

SOLUTION

```
LL dp[(1<<13)+5][15][15];
LL ans[15] ;
int n , q , r , k ;
int notok[15][15];
LL DP(int mask , int last , int left)
{
    if(mask == (( 1 << n ) -1 )) return (LL) 1;
    LL &ret = dp[mask][last][left];
    if(ret!=-1) return ret;
    ret = 0 ;
    int i ;
    for ( i = 0 ; i<n ; i++ )
    {
        if( mask & (1 << i )) continue;
        if ( left - notok[last][i+1] < 0 ) continue;
        ret += DP( mask | ( 1 << i ) , i+1 , left -
notok[last][i+1] ) ;
    }
    return ret;
}
int main()
{
    // freopen("input.txt","r",stdin);
    int cs , t , i;
    scanf("%d",&t);
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        memo(notok,0);
        printf("Case %d:\n",cs);
        scanf("%d %d %d",&n,&k,&q);
        int x , y ;
        for ( i = 1 ; i <= n ; i++ )
        {
            scanf("%d",&y);
            while(y--)
            {
                scanf("%d",&x);
                notok[i][x] = 1;
            }
        }
        memo(dp,-1);
        for ( i = 0 ; i <= n ; i++ ) {
            ans[i] = DP(0,0,i);
        }
    }
}
```

```
}  
    int idx ;  
    while(q--)  
    {  
        cin >> r ;  
        if(k==0) idx = n ;  
        else  
        {  
            idx = ( r / k ) ;  
            if(idx>n) idx = n ;  
        }  
        printf("%lld\n",ans[idx]);  
    }  
}  
return 0;  
}
```

1230 - Placing Lampposts

Time Limit: 2 second(s)

Memory Limit: 32 MB

As a part of the mission 'Beautification of Dhaka City', the government has decided to replace all the old lampposts with new expensive ones. Since the new ones are quite expensive and the budget is not up to the requirement, the government has decided to buy the minimum number of lampposts required to light the whole city.

Dhaka city can be modeled as an undirected graph with no cycles, multi-edges or loops. There are several roads and junctions. A lamppost can only be placed on junctions. These lampposts can emit light in all the directions and that means a lamppost that is placed in a junction will light all the roads leading away from it.

The 'Dhaka City Corporation' has given you the road map of Dhaka city. You are hired to find the minimum number of lampposts that will be required to light the whole city. These lampposts can then be placed on the required junctions to provide the service. There could be many combinations of placing these lampposts that will still cover all the roads. In that case, you have to place them in such a way so that the number of roads receiving light from two lampposts is maximized. (A careful thought will reveal that all the roads will get light either from one post or two posts).

Input

Input starts with an integer T (≤ 30), denoting the number of test cases.

Each case starts with a blank line. The next line contains two integers N ($1 \leq N \leq 1000$) and M ($0 \leq M < N$) which indicate the number of junctions and roads respectively. The junctions are numbered from 0 to $N-1$. Each of the next M lines contains two integers a and b ($0 \leq a, b < N, a \neq b$), which denotes that there is a road from junction a to b .

Output

For each case, print the case number, the minimum number of lampposts required to light the whole city, the number of roads that are receiving lights from two lampposts and the number of roads that are receiving light from only one lamppost.

Sample Input	Output for Sample Input
2 4 3 0 1 1 2 2 3 5 4 0 1 0 2	Case 1: 2 1 2 Case 2: 1 0 4

0 3	
0 4	

SOLUTION

```
const int NX = 1005 ;
pii dp[2][NX] ;
int cs , adj[NX][NX] , deg[NX] , n , m ;
bool mem[NX] ;
int vis[2][NX] ;
void ini()
{
    rep( i , n )
    {
        deg[i] = 0 ;
        mem[i] = 0 ;
    }
}
pii DP( int x , int isGard , int par )
{
    if( vis[isGard][x] == cs ) return dp[isGard][x];
    vis[isGard][x] = cs ;
    mem[x] = 1 ;
    pii tmp , a , b , ans = mp( isGard , 0 );
    rep( i , deg[x] )
    {
        int v = adj[x][i];
        if( v == par ) continue ;
        if( !isGard )
        {
            tmp = DP( v , 1 , x );
            ans.ff += tmp.ff ;
            ans.ss += tmp.ss ;
        }
        else
        {
            a = DP( v , 0 , x );
            b = DP( v , 1 , x );
            if( a.ff < b.ff )
            {
                ans.ff += a.ff ;
                ans.ss += a.ss ;
            }
            else if( b.ff < a.ff )
            {
                ans.ff += b.ff ;
                ans.ss += b.ss ;
                ans.ss++;
            }
        }
    }
}
```

```

    }
    else if( b.ss < a.ss )
    {
        ans.ff += a.ff ;
        ans.ss += a.ss ;
    }
    else
    {
        ans.ff += b.ff ;
        ans.ss += b.ss ;
        ans.ss++;
    }
}
}
return dp[isGard][x] = ans ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // ms( vis , -1 );
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II , m = II ;
        ini();
        rep( i , m )
        {
            int x = II , y = II ;
            adj[x][deg[x]++] = y ;
            adj[y][deg[y]++] = x ;
        }
        pii ans , a , b ;
        ans = mp( 0 , 0 );
        rep( i , n )
        {
            if(mem[i] || deg[i] == 0 )continue ;
            a = DP( i , 0 , -1 );
            b = DP( i , 1 , -1 );
            if( a.ff < b.ff )
            {
                ans.ff += a.ff ;
                ans.ss += a.ss ;
            }
            else if( b.ff < a.ff )
            {

```



```
        ans.ff += b.ff ;
        ans.ss += b.ss ;
    }
    else if( b.ss < a.ss )
    {
        ans.ff += a.ff ;
        ans.ss += a.ss ;
    }
    else
    {
        ans.ff += b.ff ;
        ans.ss += b.ss ;
    }
}
printf("Case %d: %d %d %d\n",cs,ans.ff , ans.ss , m - ans.ss );
}
return 0;
}
```

1231 - Coin Change (I)

Time Limit: 1 second(s)

Memory Limit: 32 MB

In a strange shop there are n types of coins of value $A_1, A_2 \dots A_n$. $C_1, C_2, \dots C_n$ denote the number of coins of value $A_1, A_2 \dots A_n$ respectively. You have to find the number of ways you can make K using the coins.

For example, suppose there are three coins 1, 2, 5 and we can use coin 1 at most 3 times, coin 2 at most 2 times and coin 5 at most 1 time. Then if $K = 5$ the possible ways are:

1112

122

5

So, 5 can be made in 3 ways.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing two integers n ($1 \leq n \leq 50$) and K ($1 \leq K \leq 1000$). The next line contains $2n$ integers, denoting $A_1, A_2 \dots A_n, C_1, C_2 \dots C_n$ ($1 \leq A_i \leq 100, 1 \leq C_i \leq 20$). All A_i will be distinct.

Output

For each case, print the case number and the number of ways K can be made. Result can be large, so, print the result modulo 100000007.

Sample Input	Output for Sample Input
2 3 5 1 2 5 3 2 1 4 20 1 2 3 4 8 4 2 1	Case 1: 3 Case 2: 9

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;

int k;
int n;
int a[100];
int c[100];
int dp[100][1008];

long long explore(int i, int sum)
{
    long long ans;
    int mod;
    ans = 0;

    mod = 1e8 + 7;

    if(sum == k) {
        return 1;
    }

    if(i == n) {
        return 0;
    }

    if(dp[i][sum] != -1) {
        return dp[i][sum];
    }

    for (int j = 0; j <= c[i] and (j * a[i]) + sum <= k; j++) {
        ans += explore(i+1, sum + j * a[i]);
        ans = ans % mod;
    }

    return dp[i][sum] = ans;
}

int main()
```

```
{

    int t;
    int mod;

    mod = 1e8 + 7;
    scanf("%d", &t);

    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        scanf("%d", &k);

        for (int i = 0; i < n; i++) {
            scanf("%d", a + i);
        }

        for (int i = 0; i < n; i++) {
            scanf("%d", c + i);
        }

        memset(dp, -1, sizeof dp);

        printf("Case %d: %lld\n", cs, explore(0, 0));

    }

}
```

1232 - Coin Change (II)

Time Limit: 1 second(s)

Memory Limit: 32 MB

In a strange shop there are n types of coins of value $A_1, A_2 \dots A_n$. You have to find the number of ways you can make K using the coins. You can use any coin at most K times.

For example, suppose there are three coins 1, 2, 5. Then if $K = 5$ the possible ways are:

11111

1112

122

5

So, 5 can be made in 4 ways.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing two integers n ($1 \leq n \leq 100$) and K ($1 \leq K \leq 10000$). The next line contains n integers, denoting $A_1, A_2 \dots A_n$ ($1 \leq A_i \leq 500$). All A_i will be distinct.

Output

For each case, print the case number and the number of ways K can be made. Result can be large, so, print the result modulo 100000007.

Sample Input	Output for Sample Input
2 3 5 1 2 5 4 20 1 2 3 4	Case 1: 4 Case 2: 108

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;

int k;
int n;
int a[109];
int dp[10009];

int main()
{
    int t;
    int mod;

    mod = 1e8 + 7;
    scanf("%d", &t);

    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        scanf("%d", &k);

        for (int i = 0; i < n; i++) {
            scanf("%d", a + i);
        }

        memset(dp, 0, sizeof dp);
        dp[0] = 1;
        for (int i = 0; i < n; i++) {
            for (int j = a[i]; j <= k; j++) {
                dp[j] += dp[j - a[i]];
                dp[j] = dp[j] % mod;
            }
        }

        printf("Case %d: %d\n", cs, dp[k]);
    }
}
```


1233 - Coin Change (III)

Time Limit: 2 second(s)

Memory Limit: 32 MB

In a strange shop there are n types of coins of value $A_1, A_2 \dots A_n$. $C_1, C_2, \dots C_n$ denote the number of coins of value $A_1, A_2 \dots A_n$ respectively. You have to find the number of different values (from 1 to m), which can be produced using these coins.

Input

Input starts with an integer T (≤ 20), denoting the number of test cases.

Each case starts with a line containing two integers n ($1 \leq n \leq 100$), m ($0 \leq m \leq 10^5$). The next line contains $2n$ integers, denoting $A_1, A_2 \dots A_n, C_1, C_2 \dots C_n$ ($1 \leq A_i \leq 10^5$, $1 \leq C_i \leq 1000$). All A_i will be distinct.

Output

For each case, print the case number and the result.

Sample Input	Output for Sample Input
2 3 10 1 2 4 2 1 1 2 5 1 4 2 1	Case 1: 8 Case 2: 4

SOLUTION

```
#include <stdio.h>
#include <string.h>
#include <set>
#include <algorithm>
#include <iostream>
using namespace std;
#define MAX 100005
set <int> res;
int total,n,val[MAX],coin[MAX],move[MAX];
bool money[MAX];
int main()
{
    // freopen("input.txt","r",stdin);
    int cs,t,i,j,k;
    cin>>t;
    for(cs=1;cs<=t;cs++)
    {
        //res.clear();
        memset(money,0,sizeof(money));
        money[0]=1;
        scanf("%d %d",&n,&total);
        for(i=0;i<n;i++) cin>>coin[i];
        for(i=0;i<n;i++) cin>>val[i];
        int res=0;
        for(i=0;i<n;i++)
        {
            for(k=0;k<=total;k++)
            move[k]=0;
            for(j=coin[i];j<=total;j++)
            {
                if(money[j-coin[i]] && !money[j] && move[j-
coin[i]]<val[i])
                {
                    money[j]=1;
                    move[j]=move[j-coin[i]]+1;
                    res++;
                }
            }
        }
        printf("Case %d: %d\n",cs,res);
    }
}
```

1246 - Colorful Board

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are given a rectangular board. You are asked to draw **M** horizontal lines and **N** vertical lines in that board, so that the whole board will be divided into **(M+1) x (N+1)** cells. So, there will be **M+1** rows each of which will exactly contain **N+1** cells or columns. The **yth** cell of **xth** row can be called as **cell(x, y)**. The distance between two cells is the summation of row difference and column difference of those two cells. So, the distance between **cell(x₁, y₁)** and **cell(x₂, y₂)** is

$$|x_1 - x_2| + |y_1 - y_2|$$

For example, the distance between cell (2, 3) and cell (3, 2) is $|2 - 3| + |3 - 2| = 1 + 1 = 2$.

After that you have to color every cell of the board. For that you are given **K** different colors. To make the board more beautiful you have to make sure that no two cells having the same color can have odd distance between them. For example, if you color cell (3, 5) with red, you cannot color cell (5, 8) with red, as the distance between them is 5, which is odd. Note that you can keep some color unused, but you can't keep some cell uncolored.

You have to determine how many ways to color the board using those **K** colors.

Input

Input starts with an integer **T** (≤ 20000), denoting the number of test cases.

Each case starts with a line containing three integers **M, N, K** ($0 \leq M, N \leq 19, 1 \leq K \leq 50$).

Output

For each case, print the case number and the number of ways you can color the board. The result can be large, so print the result modulo **1000000007**.

Sample Input	Output for Sample Input
4	Case 1: 1
0 0 1	Case 2: 2
0 0 2	Case 3: 2
5 5 2	Case 4: 0
5 5 1	

SOLUTION

```
const int NX = 2005 ;
const int MX = 1005 ;
const int mod = 1000000007 ;
int ncr[MX][MX] , dp[NX][ 55 ] , pw[ 55 ][MX] , n , k , m ;
int C[MX][MX];
void ini()
{
    int i , j ;
    for ( i = 0 ; i < MX ; i++ )
    {
        for ( j = 0 ; j <= i ; j++ )
        {
            if( i == 0 || j == i ) ncr[i][j] = 1 ;
            else ncr[i][j] = ( ncr[i-1][j-1] + ncr[i-1][j])%mod ;
        }
    }
}

int power(int n, int p){
    if(p == 0) return 1;
    if(p == 1) return n;
    if(pw[n][p]) return pw[n][p];
    return pw[n][p] = ((Long) power(n, p-1) * n) % mod ;
}

int DP( int cells , int colors )
{
    if( colors > cells || cells == 0 || colors == 0 ) return 0 ;
    if( colors == 1 ) return 1 ;
    int &ret = dp[ cells ][ colors ];
    if( ret != -1 ) return ret ;
    ret = 0 ;
    int i ;
    for ( i = 1 ; i <= cells ; i++ )
    {
        ret = ( ( Long ) ncr[ cells ][ i ] * DP ( cells - i , colors - 1 ) + ret
)% mod ;
    }
    return ret ;
}

int solve( int k1 , int lft )
{
    if( lft <= 0 ) return 0 ;
    int x = ( n * m )/2 ;
    int y = ( n * m + 1 ) / 2 ;
}
```

```

        if( x < k1 ) return 0 ;
        int ans = (( Long ) DP( x , k1 ) * power( lft , y ) ) % mod;
        // ans %= mod ;
        ans = ( ( Long ) ans * ncr[ k ][ k1 ]) % mod ;
        // ans %= mod ;
        return ans ;
    }
    int Ans()
    {
        int i ;
        n = II , m = II , k = II ;
        n++ , m++ ;
        if( n == 1 && m == 1 ) return k ;
        int ans = 0 ;
        for ( i = 1 ; i < k ; i++ )
        {
            ans = (solve( i , k - i ) + ans )%mod;
        }
        return ans ;
    }
    int main()
    {
        // I will always use scanf and printf
        // May be i won't be a good programmer but i will be a good human being
        //cout << ( 1 << 30 ) << endl ;
        ini();
        int cs , t = II ;
        ms( dp , -1 );
        for ( cs = 1 ; cs <= t ; cs++ )
        {
            printf("Case %d: %d\n",cs,Ans());
        }
        return 0;
    }
}

```

1252 - Maintaining Communities

Time Limit: 2 second(s)

Memory Limit: 32 MB

Tracking Communities in social networks, like facebook, twitter etc, are one of the most exiting works in the field of Artificial Intelligence now-a-days. A community is a group of people who are connected. Two persons are connected and form a community if they are directly connected or connected via some other persons. That means if person **A** and **B** are connected and **B** and **C** are also connected then **A** and **C** are connected and they all belong to the same community. It is quite obvious that, one single person can also form a community by himself.

Now you are given a description of a community where there are **n** persons and the connections between some pairs of the persons. You can assume that they do form a community and their connection is given such that if you want to find two peoples connectivity information, you will find exactly one path between them.

Now maintaining a connection between a pair requires some cost. And unfortunately, the social networking site cannot afford keeping a community which requires a cost of more than **K**. For example, say A and B and B and C are connected and **cost(A, B) = 5, cost(B, C) = 2**. Then if **K ≥ 7**, they can afford this community. Otherwise they cannot.

So, they have made a plan, and that is they will break the community. They want to break the community into multiple communities such that each community requires maintenance cost no more than **K**. But if there are too many communities people may leave the network, that's why they want the minimum number of communities. Since you are the best, they find no option but to ask you.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case starts with a line containing two integers **n** ($1 \leq n \leq 100$) and **K** ($1 \leq K \leq 100$). Each of the next **n - 1** lines contains three integers **u v w** ($1 \leq u, v \leq n, 1 \leq w \leq 100, u \neq v$) meaning that there is a connection between person **u** and **v** and the maintenance cost of this connection is **w**.

Output

For each case, print the case number and the minimum number of communities they have to maintain.

Sample Input	Output for Sample Input
2 3 1 1 2 2 2 3 2	Case 1: 3 Case 2: 2

4	12
1	2 5
2	3 10
1	4 7

SOLUTION

```
const int NX = 1005 ;
vector < int > adj[ NX ] , cost[ NX ] , chk[ NX ];
int total[ NX ] , n , k , dp[ NX ] ;
bool cmp( int a , int b )
{
    return a > b ;
}
int dfs( int x , int par )
{
    int ret = 0 , sum = 0 , idx = 0 ;
    chk[ x ].pb( 0 );
    int sz = adj[ x ].size();
    rep( i , sz )
    {
        int u = adj[ x ][ i ];
        if( u == par ) continue ;
        ret += dfs( u , x );
        sum += dp[ u ] + cost[ x ][ i ];
        chk[ x ].pb( dp[ u ] + cost[ x ][ i ] );
    }
    sort( chk[x].begin() , chk[x].end() );
    while( sum > k )
    {
        sum -= chk[x].back();
        chk[x].pop_back();
        ret++;
    }
    dp[ x ] = sum ;
    return ret ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II , k = II ;
        for ( int i = 0 ; i <= n ; i++ )
        {
            adj[i].clear();
            cost[i].clear();
            chk[i].clear();
        }
    }
}
```

```

    }
    for ( int i = 1 ; i < n ; i++ )
    {
        int x = II , y = II , c = II ;
        adj[ x - 1 ].pb( y - 1 );
        cost[ x - 1 ].pb( c );
        adj[ y - 1 ].pb( x - 1 );
        cost[ y - 1 ].pb( c );
    }
    // printf("Case %d: %d\n",cs, dfs( 0 , -1 ) + 1 );
    printf("Case %d: %d\n", cs, 1 + dfs( 1, -1));
}
return 0;
}

```


1257 - Farthest Nodes in a Tree (II)

Time Limit: 2 second(s)

Memory Limit: 32 MB

Given a tree (a connected graph with no cycles), you have to find the cost to go to the farthest node from each node. The edges of the tree are weighted and undirected.

Input

Input starts with an integer **T** (≤ 10), denoting the number of test cases.

Each case starts with an integer **n** ($2 \leq n \leq 30000$) denoting the total number of nodes in the tree. The nodes are numbered from **0** to **n-1**. Each of the next **n-1** lines will contain three integers **u v w** ($0 \leq u, v < n, u \neq v, 1 \leq w \leq 10000$) denoting that node **u** and **v** are connected by an edge whose weight is **w**. You can assume that the input will form a valid tree.

Output

For each case, print the case number in a line first. Then for each node (from **0** to **n - 1**) print the cost to go to the farthest node in a separate line.

Sample Input	Output for Sample Input
2	Case 1:
4	100
0 1 20	80
1 2 30	50
2 3 50	100
5	Case 2:
0 2 20	50
2 1 10	80
0 3 29	70
0 4 50	79
	80

Note

Dataset is huge, use faster I/O methods.

SOLUTION

```
#include <iostream>
#include <stdio.h>
#include <vector>
#include <utility>
#include <string.h>
#define vii vector < vector < pair <int, int> > >
using namespace std;
enum consts {N = 30000};
vii a;
int vis[N + 5];
int dis[N + 5];
int maxi;
int n;
int explore(int i, int d)
{
    int x;
    int y;
    int w;

    dis[i] = d;
    for (int j = 0; j < a[i].size(); j++) {
        x = a[i][j].first;
        w = a[i][j].second;
        if(!vis[x]) {
            vis[x] = 1;
            explore(x, d + w);
        }
    }
}

int routine(int start)
{
    int ans;
    memset(vis, 0, sizeof vis);
    memset(dis, 0, sizeof dis);

    maxi = -1;

    vis[start] = 1;
    explore(start, 0);

    for (int i = 0; i < n; i++) {
        if(dis[i] > maxi) {
            maxi = dis[i];
        }
    }
}
```

```

        ans = i;
    }
}
return ans;
}
int main()
{
    int t;
    int m;
    int x;
    int y;
    int w;
    int c[N+5];
    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        vii temp(n+5);
        swap(temp, a);
        for (int i = 0; i < n - 1; i++) {
            scanf("%d", &x);
            scanf("%d", &y);
            scanf("%d", &w);
            a[x].push_back(make_pair(y, w));
            a[y].push_back(make_pair(x, w));
        }

        int k1 = routine(0);
        int k2 = routine(k1);
        for(int i = 0; i < n; i++) {
            c[i] = dis[i];
        }

        int k3 = routine(k2);

        printf("Case %d:\n", cs);
        for (int i = 0; i < n; i++) {
            printf("%d\n", max(dis[i], c[i]));
        }
    }
}

```

1264 - Grouping Friends

Time Limit: 2 second(s)

Memory Limit: 32 MB

There were n friends. But some of them had problems with others. So, we can say each person has some dissatisfaction with others. And hence we introduce a new term 'Dissatisfaction Factor' and it is described as follows:

You are given all the dissatisfaction factors as d_{ij} . d_{ij} means the dissatisfaction factor according to person i towards j . If the value is negative that means i^{th} person likes j^{th} person. Positive value means i^{th} person hates j^{th} person. 0 means i^{th} person is neutral about j^{th} person. Obviously d_{ij} can be different from d_{ji} since i^{th} person may like j^{th} person, but j^{th} person may not like i^{th} person and vice versa.

Now, if you group some people, the dissatisfaction factor is the summation of all the members' dissatisfaction factors towards other people in the group. You have to group them with minimum dissatisfaction factor. You can make as many groups as you like.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing an integer n ($2 \leq n \leq 14$). Each of the next n lines contains n space separated integers. The j^{th} integer in the i^{th} line denotes d_{ij} . You can assume that $-100 \leq d_{ij} \leq 100$ and of course $d_{ii} = 0$.

Output

For each case, print the case number and the minimum dissatisfaction factor you can make after grouping them.

Sample Input	Output for Sample Input
3 3 0 2 3 -3 0 5 2 3 0 4 0 -1 -2 -3 1 0 -2 3 1 2 0 -1 2 -2 1 0 2 0 5 1 0	Case 1: -1 Case 2: -2 Case 3: 0

SOLUTION

```
const int NX = ( 1 << 14 ) + 19 ;
int dp[ NX ] , vis[ NX ] , cs ;
int inp[ 15 ][ 15 ] , sv[ NX ] ;
int DP( int mask )
{
    if( mask == 0 ) return 0 ;
    if( vis[ mask ] == cs ) return dp[ mask ] ;
    vis[ mask ] = cs ;
    int ret = 0 ;
    int i ;
    for ( i = mask ; i ; i = ( i - 1 ) & mask )
    {
        if( sv[i] >= 0 ) continue ;
        ret = min( ret , DP( mask ^ i ) + sv[i] );
    }
    dp[ mask ] = ret ;
    return dp[ mask ] ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        int n = II ;
        rep ( i , n )
        {
            rep ( j , n )
            {
                inp[i][j] = II ;
            }
        }
        int i , j , k ;
        int lim = ( 1 << n ) ;
        for ( i = 0 ; i < ( lim ) ; i++ )
        {
            sv[i] = 0 ;
            for ( j = 0 ; j < n ; j++ )
            {
                if( i & ( 1 << j ) )
                {
                    for ( k = 0 ; k < n ; k++ )
```

```

        {
            if( ( i & ( 1 << k ) ) == 0 ) continue ;
            sv[i] += inp[j][k] ;
        }
    }
}

printf("Case %d: %d\n",cs,DP( lim - 1 ) );
}

return 0;
}

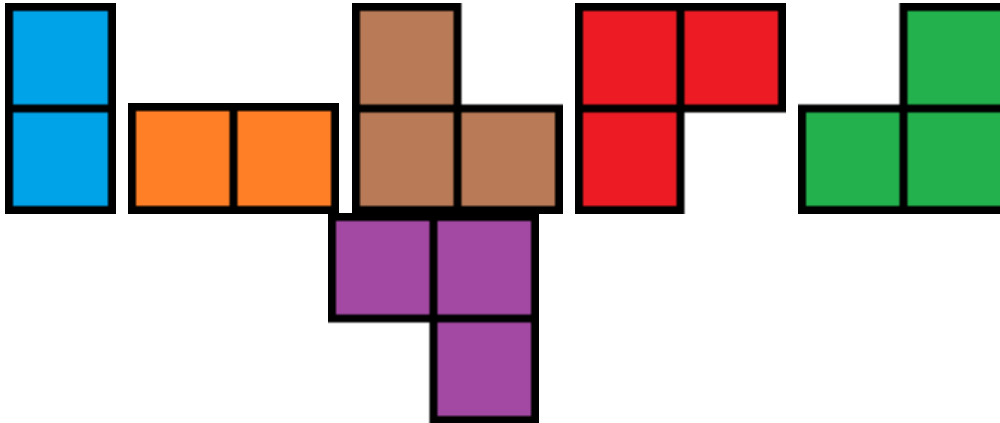
```

1270 - Tiles (II)

Time Limit: 4 second(s)

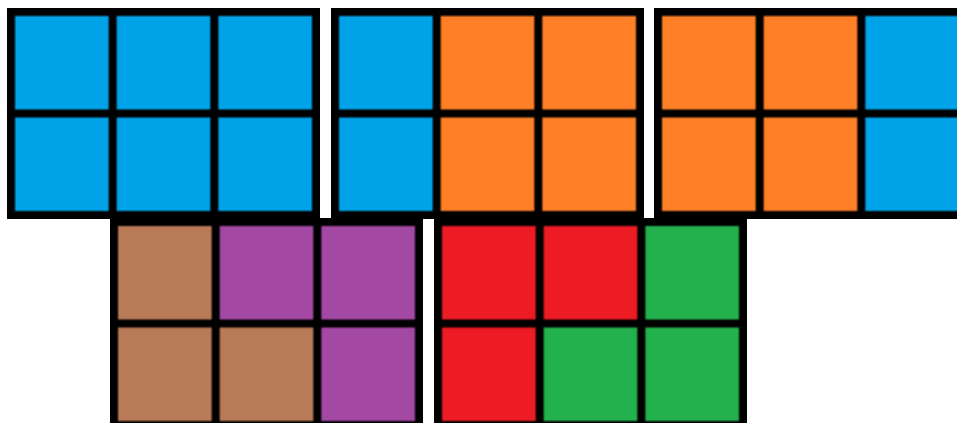
Memory Limit: 32 MB

There is an $M \times N$ board, six types of tiles are available, and each of them is infinitely many, you have to find the number of ways you can fill the board using the tiles. Two board configurations are different if at least in one cell, their colors differ. The tiles are given below:



You **cannot** rotate or flip any tile. And no cell in the board should be empty. But some cells may be broken; you can't place any part of a tile in the broken cells. And there will be at least one cell which is not broken. The tiles shouldn't overlap.

For example, a 2×3 board can be colored by 5 ways, they are:



Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing two integers: M N ($1 \leq M, N \leq 100$, $\min(M, N) \leq 8$). Each of the next M lines contains N characters forming the board. There are two types of characters. A '.' means the cell is **not** broken; a '#' means the cell is **broken**.

Output

For each case, print the case number and the number of ways the board can be colored. The number may be large, so, output the number modulo 2^{64} .

Sample Input	Output for Sample Input
3	Case 1: 5
2 3	Case 2: 2
...	Case 3: 21272
...	
2 3	
. . #	
...	
5 5	
... . .	
... . .	
... . .	
... . .	
... . .	

SOLUTION

```
const int NX = ( 1 << 8 ) + 10 ;
const int MX = 105 ;
ull dp[MX][NX] ;
int vis[MX][NX] , cs , row , col ;
char inp[MX][MX] , grid[MX][MX] ;
ull DP(int , int) ;
ull gen( int r , int c , int curmask , int nxtmask )
{
    if( c == col ) return DP( r + 1 , nxtmask );
    int cc = ( curmask >> c ) & 1 ;
    int cnc = ( curmask >> ( c + 1 ) ) & 1 ;
    int nc = ( nxtmask >> ( c ) ) & 1 ;
    int nnc = ( nxtmask >> ( c + 1 ) ) & 1 ;
    int npc = ( nxtmask >> max( 0 , c - 1 ) ) & 1 ;
    if( cc ) return gen( r , c + 1 , curmask , nxtmask );
    if( grid[r][c] == '#' ) return gen( r , c + 1 , curmask , nxtmask );
    ull ret = 0 ;
    // tile one
    if( r + 1 < row )
    {
        if( grid[r][c] == '.' && grid[r+1][c] == '.' && !cc && !nc )
        {
            ret += gen( r , c + 1 , curmask | ( 1 << c ) , nxtmask | ( 1 << c )
);
        }
    }
    // tile two
    if( c + 1 < col )
    {
        if( grid[r][c] == '.' && grid[r][c+1] == '.' && !cc && !cnc )
        {
            ret += gen( r , c + 2 , curmask | ( 1 << c ) | ( 1 << ( c + 1 ) ) ,
nxtmask );
        }
    }
    // tile five
    if( c - 1 >= 0 && r + 1 < row )
    {
        if( grid[r][c] == '.' && grid[r+1][c] == '.' && grid[r+1][c-1] == '.' &&
!cc && !nc && !npc )
        {
            ret += gen( r , c + 1 , curmask | ( 1 << c ) , nxtmask | ( 1 << c ) |
( 1 << ( c - 1 ) ) );
        }
    }
}
```

```

    }
}
if( r + 1 < row && c + 1 < col )
{
    // tile 4
    if( grid[r][c] == '.' && grid[r][c+1] == '.' && grid[r+1][c] == '.' &&
!cc && !cnc && !nc )
    {
        ret += gen( r , c + 1 , curmask | ( 1 << c ) | ( 1 << ( c + 1 ) ),
nxtmask | ( 1 << c ) );
    }
    // tile 3
    if( grid[r][c] == '.' && grid[r+1][c] == '.' && grid[r+1][c+1] == '.' &&
!cc && !nc && !nnc )
    {
        ret += gen( r , c + 1 , curmask | ( 1 << c ) , nxtmask | ( 1 << c ) |
( 1 << ( c + 1 ) ) );
    }
    // tile 6
    if( grid[r][c] == '.' && grid[r][c+1] == '.' && grid[r+1][c+1] == '.' &&
!cc && !cnc && !nnc )
    {
        ret += gen( r , c + 1 , curmask | ( 1 << c ) | ( 1 << ( c + 1 ) ) ,
nxtmask | ( 1 << ( c + 1 ) ) );
    }
}
return ret ;
}
ull DP(int r , int mask)
{
    if( r == row ) return mask == 0 ;
    ull &ret = dp[r][mask];
    int &v = vis[r][mask];
    if( v == cs ) return ret ;
    v = cs ;
    ret = gen(r , 0 , mask , 0 );
    return ret ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // ms( vis , -1 );
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {

```

```
    row = II , col = II ;
    rep( i , row ) scanf("%s",inp[i]);
    if( row < col )
    {
        rep( i , row ) rep ( j , col ) grid[j][i] = inp[i][j];
        swap( row , col );
    }
    else
    {
        rep( i , row ) rep ( j , col ) grid[i][j] = inp[i][j];
    }
    printf("Case %d: %llu\n",cs,DP(0,0));
}
return 0;
}
```

1274 - Beating the Dataset

Time Limit: 4 second(s)

Memory Limit: 32 MB

You are in a contest, and unfortunately you don't have much time. You have one problem in hand; you just glanced at the sample output and found that it just wants 'YES' or 'NO'. So, you have made another plan instead of solving the problem as you know the system very well.

For this problem, every test case is stored in a separate file. When a submission is found, the system successively runs the solution on all tests of a problem, and for each test the checking process goes as follows. The input is copied to the file input.txt. Then the solution is launched. It reads the input from the file input.txt and writes the result to the file output.txt. When it finishes, the correct answer is copied to the file answer.txt. If the contents of the files answer.txt and output.txt match, the test is assumed to be passed; otherwise, the test is not passed.

So, you decided to write a program that would operate as follows. If the folder containing the program doesn't contain the file answer.txt (i.e. the program is run on the first test), then the program outputs "YES". Otherwise, the program outputs the contents of the file answer.txt. And before the contest, the sizes of the data files are given to you.

And it's clear that the size of the file with the answer "YES" is 3 bytes, the size of the file with the answer "NO" is 2 bytes, and all the variants of the order of tests are equally probable. Now you want to calculate the average number of tests that your solution won't pass.

Input

Input starts with an integer T (≤ 10), denoting the number of test cases.

Each case starts with a line containing two integers n ($1 \leq n \leq 5000$) and s ($2n \leq s \leq 3n$) where n denotes the number of data sets and s denotes the total size of the answer files.

Output

For each case, print the case number and the average number of tests your solution won't pass. Error less than 10^{-6} will be ignored.

Sample Input	Output for Sample Input
4	Case 1: 2
3 7	Case 2: 1
1 2	Case 3: 0
1 3	Case 4: 2.5000000000
4 10	

Note

For the first case, one of the three answers is "YES" and two answers are "NO". If the order of tests is "YES-NO-NO", then your solution won't pass the second test only; if the order is "NO-YES-NO", then it will pass none of the tests; if the order is "NO-NO-YES", the solution won't pass the first and the third tests.

SOLUTION

```
using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
const int MN = 5005;
double dp[2][MN][2];
void fill_dp(set<pair<int, int> > &queries, map<pair<int, int>, double >
&answers) {
    memset(dp, 0, sizeof dp);
    int cur = 1, prev = 0;
    for (int n = 1; n < MN; ++n) {
        memset(dp[cur], 0, sizeof dp[cur]);
        for (int y = 0; y <= n; ++y) {
            double p_no = (n - y) / (double) n;
            double p_yes = y / (double) n;
            for (int a = 0; a < 2; ++a) {
                if (y < n)
                    dp[cur][y][a] += (dp[prev][y][0] + (a != 0)) * p_no;
                if (y > 0)
                    dp[cur][y][a] += (dp[prev][y - 1][1] + (a != 1)) * p_yes;
            }
            if (queries.count(make_pair(n, y)))
                answers[make_pair(n, y)] = dp[cur][y][1];
        }
        swap(cur, prev);
    }
}

void read_in(set<pair<int, int> > &queries, vector<pair<int, int> > &order) {
    int n, m;
    cin >> n >> m;
    int y = m - n - n;
    queries.insert(make_pair(n, y));
    order.push_back(make_pair(n, y));
}

int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int tc; cin >> tc;
    set<pair<int, int> > queries;
    vector<pair<int, int> > order;
    for (int i = 0; i < tc; ++i)
        read_in(queries, order);
    map<pair<int, int>, double> answers;
    fill_dp(queries, answers);
    for (int i = 0; i < tc; ++i)
```

```
    printf("Case %d: %.10lf\n", i + 1, answers[order[i]]);  
    return 0;  
}
```


1277 - Looking for a Subsequence

Time Limit: 2 second(s)

Memory Limit: 32 MB

Given a sequence of integers $S = \{a_1, a_2, a_3 \dots a_n\}$ and an integer m , you have to find a subsequence of S containing m elements, $P = \{b_1, b_2, b_3 \dots b_m\}$ such that $(b_1 < b_2 < \dots < b_m)$. If there are several subsequences possible then you should find the one where b_1 is leftmost. If there is still a tie, then check for the one where b_2 is leftmost and so on.

For example, let the sequence be, 8 7 5 6 5 1 2 7 and $m = 2$. Then (1, 2), (5, 6), (5, 7), (1, 7) ... etc can be solutions. But we are looking for (5, 6).

Input

Input starts with an integer T (≤ 12), denoting the number of test cases.

Each case starts with a line containing two integers n ($1 \leq n \leq 10^5$) and q ($1 \leq q \leq 10$) where q denotes the total number of queries. The next line contains n space separated integers denoting $a_1, a_2 \dots a_n$ respectively. Each of the next q lines contains an integer m ($1 \leq m \leq n$). You can assume that $-10^8 \leq a_i \leq 10^8$.

Output

For each case, print the case number in a line. Then for each query print a single line containing the desired subsequence or 'Impossible' if there is no such subsequence. Insert a single space between two integers of a subsequence.

Sample Input	Output for Sample Input
3	Case 1:
6 3	Impossible
3 4 1 2 3 6	1 2 3 6
6	3 4
4	Case 2:
2	2 4 6
6 2	Impossible
2 4 6 1 3 5	Case 3:
3	8
4	5 6
8 5	5 6 7
8 7 5 6 5 1 2 7	Impossible
1	Impossible
2	
3	
4	
5	

Note

Dataset is huge, use faster I/O methods.

SOLUTION

```
const int NX = 2e5 + 10 ;
const Long INF = 2e18 + 10 ;
const Long add = 1e9 + 10 ;
Long inp[NX] , dp[NX] , L[NX] ;
vector < int > pos[NX] ;
int n , q ;
int query[20];
int vis[20];
vector < Long > ans[20];
bool solve( int lim )
{
    bool ok = 0 ;
    Long prv ;
    rep( i , n )
    {
        if( L[i] >= lim )
        {
            if( ok && inp[i] <= prv ) continue ;
            if( ok ) printf(" ");
            printf("%lld",inp[i]);
            prv = inp[i] ;
            ok = 1 ;
            lim--;
        }
        if(lim == 0 ) break ;
    }
    if( ok == 0 ) return 0 ;
    puts("");
    return 1 ;
}

int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // ms( vis , -1 );
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II , q = II ;
        rep( i , n )
        {
            inp[i] = LL ;
            dp[i] = INF ;
        }
    }
}
```

```

    }
    int low , i , j ;
    dp[0] = -INF ;
    dp[n] = INF ;
    for ( i = n - 1 ; i >= 0 ; i-- )
    {
        low = lower_bound( dp , dp + n + 1 , -inp[i] ) - dp ;
        dp[low] = -inp[i];
        L[i] = low ;
    }
    printf("Case %d:\n",cs);
    rep( i , q )
    {
        int x = II ;
        if( solve(x) == 0 ) printf("Impossible\n");
    }
}
return 0;
}

```

1283 - Shelving Books

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are a librarian. You keep the books in a well organized form such that it becomes simpler for you to find a book and even they look better in the shelves.

One day you get n new books from one of the library sponsors. And unfortunately they are coming to visit the library, and of course they want to see their books in the shelves. So, you don't have enough time to shelve them all in the shelf in an organized manner since the heights of the books may not be same. But it's the question of your reputation, that's why you have planned to shelve them using the following idea:

- 1) You will take one book from the n books from left.
- 2) You have exactly one shelf to organize these books, so you may either put this book in the left side of the shelf, right side of the shelf or you may not put it in the shelf. There can already be books in the left or right side. In such case, you put the book with that book, but you don't move the book you put previously.
- 3) Your target is to put the books in the shelf such that from left to right they are sorted in **non-descending** order.
- 4) Your target is to put as many books in the shelf as you can.

You can assume that the shelf is wide enough to contain all the books. For example, you have 5 books and the heights of the books are 3 9 1 5 8 (from left). In the shelf you can put at most 4 books. You can shelve 3 5 8 9, because at first you got the book with height 3, you stored it in the left side of the shelf, then you got 9 and you put it in the right side of the shelf, then you got 1 and you ignored it, you got 5 you put it in the left with 3. Then you got 8 and you put it in left or right. You can also shelve 1 5 8 9 maintaining the restrictions.

Now given the heights of the books, your task is to find the maximum number of books you can shelve maintaining the above restrictions.

Input

Input starts with an integer T (≤ 200), denoting the number of test cases.

Each case starts with a line containing an integer n ($1 \leq n \leq 100$). Next line contains n space separated integers from $[1, 10^5]$. The i^{th} integer denotes the height of the i^{th} book from left.

Output

For each case, print the case number and the maximum number of books that can be shelved.

Sample Input	Output for Sample Input
2 5 3 9 1 5 8 8 121 710 312 611 599 400 689 611	Case 1: 4 Case 2: 6

SOLUTION

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
#define MAX 103
#define INF 1<<29
#define memo(a,b) memset(a,b,sizeof(a))
#define FOR(a,b,c) for(a=b;a<=c;a++)
using namespace std;
bool vis[MAX][MAX][MAX];
int dp[MAX][MAX][MAX],len,a[MAX];
#define S(V) V[l][r][idx]
int DP(int l,int r,int idx)
{
    if(idx>len) return 0;
    if(S(vis)) return S(dp);
    S(vis)=1;
    int left=0,right=0,central;
    if(a[l]<=a[idx] && a[idx]<=a[r])
    {
        left=1+DP(idx,r,idx+1);
        right=1+DP(l,idx,idx+1);
    }
    central=DP(l,r,idx+1);
    return S(dp)=max(central,max(left,right));
}
int main()
{
    int cs,t,i;
    scanf("%d",&t);
    FOR(cs,1,t)
    {
        scanf("%d",&len);
        FOR(i,1,len) scanf("%d",&a[i]);
        a[0]=0;
        a[len+1]=INF;
        memo(vis,0);
        printf("Case %d: %d\n",cs,DP(0,len+1,1));
    }
}
```

1287 - Where to Run

Time Limit: 2 second(s)

Memory Limit: 32 MB

Last night you robbed a bank but couldn't escape and when you just got outside today, the police started chasing you. The city, where you live in, consists of some junctions which are connected by some bidirectional roads.

Since police is behind, you have nothing to do but to run. You don't know whether you would get caught or not, but if it is so, you want to run as long as you can. But the major problem is that if you leave a junction, next time you can't come to this junction, because a group of police wait there for you as soon as you left it, while some other keep chasing you.

That's why you have made a plan to fool the police as longer time as possible. The plan is, from your current junction, you first find the number of junctions which are safe (no police are there) and if you go to one of them; you are still able to visit all the safe junctions (in any order) maintaining the above restrictions. You named them 'Elected Junction' or **EJ**. If there is no such junction; you stop running, because you lose your mind thinking what to do, and the police catch you immediately.



But if there is at least one EJ, you can either fool around the police by staying in the current junction for 5 minutes (actually you just hide there, so the police lose your track thinking which road you might have taken), or you can choose to go to any EJ. The probability of choosing to stay in the current junction or to go to each of the EJ is equal. For example, from the current junction you can go to three EJs, that means the probability of staying in the current junction is $\frac{1}{4}$ or the probability to go to any of the EJ is $\frac{1}{4}$ since you have four options (either stay in the current junction or go to any of the three junctions).

You can fool the police (by hiding) multiple times in a city, but of course the above conditions should be satisfied. And you have decided not to stop in the middle of any road, because you have the fear that, if you stop in the middle of any road, then the police would surround you from both ends.

Now, given the map of the city and the required time for you to travel in each road of the map; you have to find the expected time for the police to catch you.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a blank line. Next line contains two integers n ($1 \leq n \leq 15$) denoting the number of junctions and m , denoting the number of roads in the city. The junctions are numbered from 0 to $n - 1$.

Each of the next m lines contains three integers $u \ v \ w$ ($0 \leq u, v < n$, $0 < w \leq 100$, $u \neq v$) meaning that there is a road between junction u and v and you need w minutes to travel in the road. Your home is in junction 0 and you are initially in your home. And you may safely assume that there can be at most one road between a pair of junctions.

Output

For each case, print the case number and the expected time in minutes. Errors less than 10^{-6} will be ignored.

Sample Input	Output for Sample Input
3 3 2 0 1 3 1 2 3 4 6 0 1 75 0 2 86 0 3 4 1 2 1 1 3 53 2 3 10 5 5 0 1 10 1 2 20 2 3 30 1 3 20 3 4 10	Case 1: 16 Case 2: 106.8333333333 Case 3: 90

Note

For the 3rd case, initially you are in junction 0, and you can either stay here for 5 minutes, or you can move to 1. The probability of staying in 0 is **0.5** and the probability of going to junction 1 is also **0.5**. Now if you are in junction 1, either you can stay here for 5 minutes or you can move to junction 2. From junction 1, you cannot move to junction 3, because if you go to junction 3, you can move to junction 2 or junction 4, but if you go to 2, you cannot visit junction 4 (since police would have occupied junction 3), and if you go to junction 4 from 3, you cannot visit junction 2 for the same reason. So, from 1, junction 2 is the only **EJ**, but junction 3 is not.

SOLUTION

```
const int NX = 1 << 16 ;
int adj[ 16 ][ 16 ] , deg[ 16 ] , cost[ 16 ][ 16 ] , cs , n , m ;
int vis[ NX ][ 16 ] , vis2[ NX ][ 16 ] ;
double dp[ NX ][ 16 ] ; bool dp2[ NX ][ 16 ] ;
bool dfs( int mask , int lst )
{
    if( mask == ( 1 << n ) - 1 ) return true ;
    int &v = vis2[mask][lst] ;
    bool &r = dp2[mask][lst] ;
    if( v == cs ) return r ;
    v = cs ;
    r = false ;
    int u ;
    rep( i , deg[lst] )
    {
        int u = adj[lst][i];
        if( mask & ( 1 << u ) ) continue ;
        r |= dfs( mask | ( 1 << u ) , u );
    }
    return r ;
}
double DP( int mask , int lst )
{
    if( mask == ( 1 << n ) - 1 ) return 0.0 ;
    double &ret = dp[mask][lst];
    int &v = vis[mask][lst];
    if( v == cs ) return ret ;
    v = cs ;
    ret = 5.0 ;
    double cnt = 0 ;
    rep( i , deg[lst] )
    {
        int u = adj[lst][i];
        if( mask & ( 1 << u ) ) continue ;
        if( dfs( mask | ( 1 << u ) , u ) == 0 ) continue ;
        cnt++;
        ret += cost[lst][i] + DP( mask | ( 1 << u ) , u );
    }
    if( cnt ) ret *= ( 1 / cnt );
    else ret -= 5.00 ;
    return ret ;
}
int main()
```

```

{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // ms( vis , -1 );
    // cout << fixed << setprecision(10);
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II , m = II ;
        ms( deg , 0 );
        rep( i , m )
        {
            int x = II , y = II , c = II ;
            adj[x][deg[x]] = y ;
            cost[x][deg[x]++] = c ;
            adj[y][deg[y]] = x ;
            cost[y][deg[y]++] = c ;
        }
        printf("Case %d: %lf\n",cs,DP(1,0));
        //cout << DP( 1 , 0 ) << endl ;
    }
    return 0;
}

```

1295 - Lighting System Design

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are given the task to design a lighting system for a huge conference hall. After doing a lot of calculation & sketching, you have figured out the requirements for an energy-efficient design that can properly illuminate the entire hall. According to your design, you need lamps of n different power ratings. For some strange current regulation method, all the lamps need to be fed with the same amount of current. So, each category of lamp has a corresponding voltage rating. Now, you know the number of lamps and cost of every single unit of lamp for each category. But the problem is that you are to buy equivalent voltage sources for all the lamp categories. You can buy a single voltage source for each category (each source is capable of supplying to infinite number of lamps of its voltage rating) and complete the design. But the accounts section of your company soon figures out that they might be able to reduce the total system cost by eliminating some of the voltage sources and replacing the lamps of that category with higher rating lamps. Certainly you can never replace a lamp by a lower rating lamp as some portion of the hall might not be illuminated then. You are more concerned about money-saving rather than energy-saving. Find the minimum possible cost to design the system.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing an integer n ($1 \leq n \leq 1000$). Each of the next n lines contains four integers V_i , K_i , C_i and L_i ($1 \leq V_i \leq 10^5$, $1 \leq K_i \leq 1000$, $1 \leq C_i \leq 10$, $1 \leq L_i \leq 100$). Here the integers in i^{th} line have the following meaning.

1. V_i means the voltage rating,
2. K_i means the cost of a voltage source of this category,
3. C_i means the cost of a lamp of this category and
4. L_i means the number of required lamps of this category.

You can assume that the voltage rating for the categories will be distinct.

Output

For each case, print the case number and the minimum possible cost to design the system.

Sample Input	Output for Sample Input
1 3 100 500 10 20 120 600 8 16 220 400 7 18	Case 1: 778

SOLUTION

```
const int NX = 1005 ;
const int INF = 1 << 29 ;
int dp[NX] , sum[NX] ;
struct abc
{
    int v , k , c , l ;
}inp[NX];
bool cmp( abc A , abc B )
{
    return A.v < B.v ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // ms( vis , -1 );
    // cout << fixed << setprecision(10);
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        int n = II ;
        For( i , n )
        {
            inp[i].v = II , inp[i].k = II , inp[i].c = II , inp[i].l = II
;
        }
        sort ( inp + 1 , inp + n + 1 , cmp );
        int i , j ;
        for ( i = 1 ; i <= n ; i++ )
        {
            dp[i] = INF ;
            sum[i] = sum[i-1] + inp[i].l ;
            for ( j = 0 ; j < i ; j++ )
            {
                dp[i] = min( dp[i] , dp[j] + inp[i].k + ((sum[i] -
sum[j])*inp[i].c));
            }
        }
        printf("Case %d: %lld\n",cs,dp[n]);
    }
    return 0;
}
```


1298 - One Theorem, One Year

Time Limit: 2 second(s)

Memory Limit: 32 MB

A number is **Almost-K-Prime** if it has exactly **K** prime numbers (not necessarily distinct) in its prime factorization. For example, $12 = 2 * 2 * 3$ is an **Almost-3-Prime** and $32 = 2 * 2 * 2 * 2 * 2$ is an **Almost-5-Prime** number. A number **X** is called **Almost-K-First-P-Prime** if it satisfies the following criterions:

1. **X** is an **Almost-K-Prime** and
2. **X** has **all and only** the first **P** ($P \leq K$) primes in its prime factorization.

For example, if **K=3** and **P=2**, the numbers $18 = 2 * 3 * 3$ and $12 = 2 * 2 * 3$ satisfy the above criterions. And $630 = 2 * 3 * 3 * 5 * 7$ is an example of **Almost-5-First-4-Prime**.

For a given **K** and **P**, your task is to calculate the summation of $\Phi(X)$ for all integers **X** such that **X** is an **Almost-K-First-P-Prime**.

Input

Input starts with an integer **T** (≤ 10000), denoting the number of test cases.

Each case starts with a line containing two integers **K** ($1 \leq K \leq 500$) and **P** ($1 \leq P \leq K$).

Output

For each case, print the case number and the result modulo 1000000007.

Sample Input	Output for Sample Input
3	Case 1: 10
3 2	Case 2: 816
5 4	Case 3: 49939643
99 45	

Note

1. In mathematics $\Phi(X)$ means the number of relatively prime numbers with respect to **X** which are smaller than **X**. Two numbers are relatively prime if their GCD (Greatest Common Divisor) is 1. For example, $\Phi(12) = 4$, because the numbers that are relatively prime to 12 are: 1, 5, 7, 11.
2. For the first case, **K = 3** and **P = 2** we have only two such numbers which are **Almost-3-First-2-Prime**, $18=2*3*3$ and $12=2*2*3$. The result is therefore, $\Phi(12) + \Phi(18) = 10$.

SOLUTION

```
const int NX = 510 ;
const int MX = 1e5 ;
const Long mod = 1000000007 ;
Long prime[NX];
bool check[MX];
Long dp[NX][NX] ;
void ini()
{
    int cnt = 1 ;
    prime[cnt++] = 2 ;
    int i , j ;
    for ( i = 3 ; cnt < NX ; i += 2 )
    {
        if( check[i] == 0 )
        {
            prime[cnt++] = i ;
            for ( j = i * i ; j < MX ; j += ( 2 * i ) ) check[j] = 1 ;
        }
    }
    dp[1][1] = 1 ;
    for ( i = 2 ; i <= 500 ; i++ )
    {
        dp[1][i] = 2 * dp[1][i-1] % mod ;
    }
    for ( i = 2 ; i <= 500 ; i++ )
    {
        for ( j = i ; j <= 500 ; j++ )
        {
            Long add = ( prime[i] * dp[i][j-1] ) ;
            add += ( dp[i-1][j-1] * ( prime[i] - 1 ) ) ;
            add %= mod ;
            dp[i][j] = add ;
        }
    }
}

int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    ini();
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
```

```
        int k = II , p = II ;  
        printf("Case %d: %lld\n",cs,dp[p][k]);  
    }  
    return 0;  
}
```

1299 - Fantasy Cricket

Time Limit: 2 second(s)

Memory Limit: 32 MB

ICC World Cup 2011 has just finished. During the world cup, lots of cricket fans were playing an online game named "Fantasy Cricket".



The score board of fantasy cricket looks like the following image. After each match of the world cup the score board of fantasy cricket updates.

Rank		Manager	Team Name	Total
1	.▲	Mind the Gap	Mind the Gap XI	12384
2	.▼	Old Man	Old Man XI	12344
3	.▲	Legend	Legend XI	11611
4	-.▼	Far Cry	Far Cry XI	11221
5		Accepted	Accepted XI	10111
6	▶▲	WA	WA XI	10001

Figure 1

Each player plays a role of a manager here. In the rank list there is a symbol besides each manger. There are three kinds of symbols. These are explained in the following table.

Symbol	Explanation	ASCII Representation
▲	The rank of the player has upgraded after last match, i.e. if the current rank of the player is K , the rank of the player before the last match was greater than K .	U

	The rank of the player has downgraded after the last match, i.e. if the current rank of the player is K , the rank of the player before the last match was less than K .	D
	The rank of the player has not changed after the last match, i.e. if the current rank of the player is K , the rank of the player before the last match was also K .	E

You will be given such a score board after a particular match. Can you determine any possible valid ordering of the players exactly before that round? For this problem you have to print the number of possible ordering before the last match.

Here is an example -

Rank		Manager
1		A
2		B
3		C
4		D

Figure 2

Possible Previous Order 1	Possible Previous Order 2
B	B
A	D
D	A
C	C

Figure 3

For this rank (figure 2), only two different ordering are possible (figure 3) before the last match which comply the current ordering with the symbols.

Name of the managers are not important for this problem. So, for a scoreboard, you will be given a sequence of ASCII representation of the symbols (stated above), i.e. you will be given a string which only contains 'U', 'D' and 'E'.

Input

Input starts with an integer **T** (≤ 300), denoting the number of test cases.

Each case starts with a line containing a string. The length of the string will be between **1** and **1000**. The string will contain characters from {'U', 'D', 'E'}.

Output

For each case, print the case number and the number of possible orderings modulo **1000000007**.

Sample Input	Output for Sample Input
3 UDUD EEE DU	Case 1: 2 Case 2: 1 Case 3: 0

SOLUTION

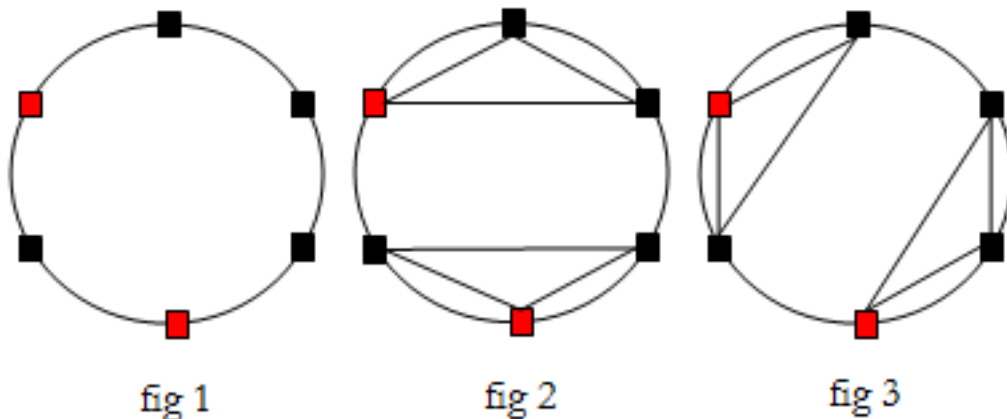
```
int X[] = {-1, -1, -1, 0, 1, 1, 1, 0};
int Y[] = {-1, 0, +1, 1, 1, 0, -1, -1};
char str[1002];
int n;
ll dp[1002][1002];
ll rec(int pos, int prevU){
    if(prevU<0)return 0;
    if(pos==n)return (prevU==0);
    ll &ret=dp[pos][prevU];
    if(ret!=-1)return ret;
    ret=0;
    if(str[pos]=='E')return ret=rec(pos+1,prevU);
    else if(str[pos]=='U'){
        ret=(prevU*rec(pos+1,prevU))%mod;
        ret+=rec(pos+1,prevU+1);ret%=mod;
    }
    else{
        ret=(prevU*rec(pos+1,prevU-1))%mod;
        ret+=rec(pos+1,prevU);ret%=mod;
        ret=(ret*prevU)%mod;
    }
    return ret;
}
int main() {
    #ifdef LOCAL
    open
    #endif // LOCAL
    int test;
    read(test);
    FOR(C, 1, test) {
        out(C);
        scanf("%s",str);
        n=strlen(str);
        for(int i=0;i<=n;i++)for(int j=0;j<=n;j++)dp[i][j]=-1;
        printf("%lld\n",rec(0,0));
    }
    return 0;
}
```

1302 - Independent Attacking Zones

Time Limit: 2 second(s)

Memory Limit: 32 MB

A common technique used by invading armies is to surround a city instead of directly entering it. The armies divided themselves into platoons having bases in a circular fashion around the city. To take internal control of the city, platoons are grouped in three to cover triangular regions. It is a policy of the General to ensure that no two triangular regions overlap. Unfortunately, the process is made a bit trickier because there are two types of armies in the invading force. The two different armies are known as Red Army and Black Army. A platoon consists of one type of army. While the Black Army has clear intention to serve the General but the Red ones might betray if they get an opportunity. It is decided that every triangular group will consist of at most one Red Army Platoon so that the Red ones cannot dominate in any assignment. Suppose we have 6 platoons (4 black and 2 red) as shown in fig 1. We have only two valid arrangements, shown in fig 2 and fig 3.



You will be given the number of platoons, their positions and colors. You have to find the number of possible configurations such that every platoon is part of exactly one group and also meets the above restrictions.

Input

Input starts with an integer T (≤ 125), denoting the number of test cases.

Each case starts with a line containing a non empty string. Each character of the string is either an 'R' or 'B'. The string gives the position of the platoons in clockwise order. 'R' indicates red and 'B' indicates black. The starting position is arbitrarily chosen. So, the example above may be represented by any of the following: 'RBBBBB', 'BBBBRB', 'BBRBRB', 'BRBRBB', 'RBRBBB' or 'BRB BBR'. You can assume that the length of the string is a multiple of 3 and not great than 70.

Output

For each case, print the case number and the total number of valid configurations.

Sample Input	Output for Sample Input
3 RBBBBRB BRBRBB BBBBBBBBB	Case 1: 2 Case 2: 2 Case 3: 12

SOLUTION

```
const int NX = 80 ;
Long dp[NX][NX] ;
int vis[NX][NX] , cs ;
char str[NX];
Long DP( int l , int r )
{
    if( l > r ) return 1 ;
    Long &ret = dp[l][r];
    int &v = vis[l][r];
    if( v == cs ) return ret ;
    v = cs ;
    ret = 0 ;
    int i , j ;
    for ( i = l + 1 ; i < r ; i++ )
    {
        for ( j = i + 1 ; j <= r ; j++ )
        {
            int red = 0 ;
            if( str[l] == 'R' ) red++;
            if( str[i] == 'R' ) red++;
            if( str[j] == 'R' ) red++;
            if( red < 2 )
                ret += ( DP(l+1,i-1) * DP( i + 1 , j - 1 ) * DP( j + 1 , r ) );
        }
    }
    return ret ;
}

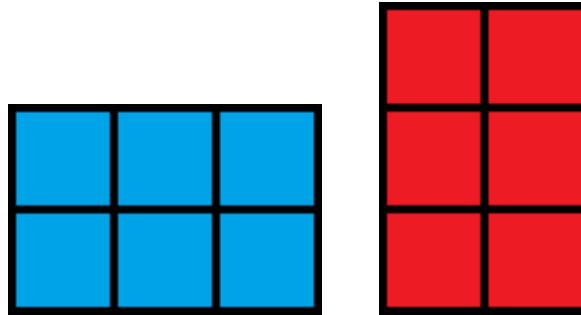
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // ms( vis , -1 );
    // cout << fixed << setprecision(10);
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        scanf("%s",str);
        printf("Case %d: %lld\n",cs,DP(0,strlen(str)-1));
    }
    return 0;
}
```


1310 - Tiles (III)

Time Limit: 2 second(s)

Memory Limit: 32 MB

There is an $M \times N$ board, two types of tiles are available, and each of them is infinitely many, you have to place maximum number of non-overlapping tiles in the board. The tiles are given below:



You **cannot** rotate or flip any tile. Some cells in the board may be broken; you can't place any part of a tile in the broken cells.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing two integers: M N ($2 \leq M \leq 8$, $2 \leq N \leq 100$). Each of the next M lines contains N characters forming the board. There are two types of characters. A '.' means the cell is **not** broken; a '#' means the cell is **broken**.

Output

For each case, print the case number and maximum number of tiles that can be placed in the board.

Sample Input	Output for Sample Input
3 2 3 2 3 ..# ... 5 6 .#....#. ..#....	Case 1: 1 Case 2: 0 Case 3: 3

SOLUTION

Can't find any!

1312 - Corrupted Tax Department

Time Limit: 2 second(s)

Memory Limit: 32 MB

There is a strange country, for simplicity we name the country as 'You Know Which', you will know the country after reading the following. The Income Tax department of the country is so corrupted that people use to say that after paying the tax, they are stolen by 'You Know Who'. So, the country is considered as an underdeveloped but leading corrupted countries for a few decades (may be we will see a century or even millennium!). So, people are dissatisfied with the Govt. but still all the taxes paid by general people are taken by 'You Know Who'.

So, some of the talented ACM solvers have realized the fact and they took some initiatives to stop the corruption in the income tax department, and want to save the department from the mysterious 'You Know Who'. In the income tax department, there is an officer who is the head of the department. Every other officer is either subordinate of him or any other officer. But any officer is direct or indirect subordinate of the head. An officer is a subordinate of exactly one other officer.

As the ACM-ers don't know which officers (or the head) are corrupted, so, they planned that every officer or the head should work with his direct subordinate and these two should make a group. The fact is that since they are not in same level, it would be tough for them to join 'You Know Who'. It may be possible that some members cannot form a group, so the ACM-ers want to form maximum number of groups. And they also want to find the number of ways they can form maximum groups.

Input

Input starts with an integer T (≤ 30), denoting the number of test cases.

Each case starts with a line containing an integer n ($1 \leq n \leq 10000$), where n denotes the number of officers (including the head). The head is identified by 1 and others are identified from 2 to n . Each of the next n lines contains the id of the person, the number of subordinates of this person, and the list of ids of the subordinates of this person separated by spaces. In the list no id is given more than once, and you can assume that the given input follows the restrictions described above.

Output

For each case, print the case number, the maximum number of groups and the number of ways to make maximum groups modulo 10007.

Sample Input	Output for Sample Input
1 5 1 2 2 4 4 1 5	Case 1: 2 3

2	1	3
3	0	
5	0	

Note

Dataset is huge, use faster I/O methods.

SOLUTION

```
const int mod = 10007 ;
const int NX = 10000 + 10 ;
int MaxGrp[ NX ][ 2 ] , ways[ NX ][ 2 ] , n ;
bool vis[ NX ];
vector < int > adj[ NX ] ;
void ini()
{
    rep( i , n + 4 )
    {
        adj[i].clear();
        vis[i] = 0 ;
    }
}
void dfs( int x )
{
    vis[x] = true ;
    MaxGrp[x][0] = MaxGrp[x][1] = 0 ;
    ways[x][0] = ways[x][1] = 1 ;
    int sz = adj[x].size();
    rep( i , sz )
    {
        int u = adj[x][i];
        if( vis[u] ) continue ;
        dfs( u );
        int mx , tmpmul , svmaxgrp[2] , svways[2];
        mx = max( MaxGrp[u][0] , MaxGrp[u][1] );
        tmpmul = MaxGrp[u][0] > MaxGrp[u][1] ? ways[u][0] : ways[u][1];
        if( MaxGrp[u][0] > 0 && MaxGrp[u][0] == MaxGrp[u][1] ) tmpmul =
ways[u][0] + ways[u][1];
        svmaxgrp[0] = MaxGrp[x][0] + mx ;
        svways[0] = ways[x][0] * tmpmul ;
        svmaxgrp[1] = max( MaxGrp[x][1] + mx , MaxGrp[x][0] + MaxGrp[u][0] + 1
);
        svways[1] = MaxGrp[x][1] + mx > MaxGrp[x][0] + MaxGrp[u][0] + 1 ?
ways[x][1] * tmpmul : ( ways[x][0] * ways[u][0] );
        if( MaxGrp[x][1] && MaxGrp[x][1] + mx == MaxGrp[x][0] + MaxGrp[u][0] + 1
) svways[1] = ways[x][1] * tmpmul + ( ways[x][0] * ways[u][0] );
        for( int j = 0 ; j < 2 ; j++ )
        {
            ways[x][j] = svways[j] % mod ;
            MaxGrp[x][j] = svmaxgrp[j];
        }
    }
}
```



```

}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II ;
        ini();
        rep( i , n )
        {
            int u = II ;
            int cnt = II ;
            rep( j , cnt )
            {
                int v = II ;
                adj[u].pb(v);
            }
        }
        dfs( 1 );
        int grp = max( MaxGrp[1][0] , MaxGrp[1][1] );
        int way = MaxGrp[1][0] > MaxGrp[1][1] ? ways[1][0] : ways[1][1];
        if( MaxGrp[1][0] > 0 && MaxGrp[1][0] == MaxGrp[1][1] ) way = ways[1][0]
+ ways[1][1];
        way %= mod ;
        printf("Case %d: %d %d\n",cs,grp,way);
    }
    return 0;
}

```

1316 - A Wedding Party

Time Limit: 3 second(s)

Memory Limit: 32 MB

We all know that we have a big and exciting wedding party ahead. So we made a plan to go to buy a gift for the wedding party. We all gathered at a place and we were just about to buy the gift. Unfortunately, we find that we have a 'Team Practice Contest' ahead. Now before going to the contest we have to buy the gift. As time is too short we will try to buy the gift on the way to the contest. We will try to visit as many shops as possible. The city map is represented by a graph with N nodes and M edges. N nodes represent the N junctions and M edges represent the **M**unidirectional roads connecting the cities. Every road has a cost which represents the required time to use the road. The contest is running at junction $N-1$ and we will start our journey at junction 0 . And there are exactly S shops located at different junctions.

Now given the location of the shops you have to find the route from junction 0 to junction $N-1$ which will visit maximum number of shops with minimum time (first maximize the number of shops then minimize the time to visit them). We can visit a junction more than once.

Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each case begins with three non negative integers N ($2 \leq N \leq 500$), M ($1 \leq M \leq 10000$) S ($0 \leq S \leq 15$). Next line contains S integers denoting the shop locations. Each of the next M lines contains three integers u, v, w ($0 < u, v < N, u \neq v, 1 \leq w \leq 100$) denoting a road from u to v with cost w .

Output

For each case of input you have to print the case number and two integers representing maximum number of shops we can visit in the way and the minimum time required to reach junction $N-1$ after visiting maximum number of shops. If we cannot attend the contest, print "**Impossible**". See samples for more details.

Sample Input	Output for Sample Input
2 4 4 4 0 1 2 3 0 1 10 1 3 30 0 2 30 2 3 5 4 4 4 0 1 2 3 0 1 10	Case 1: 3 35 Case 2: Impossible

3	1	30	
0	2	30	
3	2	5	

SOLUTION

```
const int NX = 505 ;
const int INF = 1 << 29 ;
int deg[NX] , adj[NX][NX] , cost[NX][NX] , n , m , cs , vis[ 17 ][ 1 << 17 ] ,
shop[ 20 ] ;
int dis[ NX ] , mem[ NX ] , possible[ 18 ][ 18 ] , Isshop[ NX ] , s , add[ 18 ][
18 ] ;
pii dp[ 17 ][ 1 << 17 ] ;
void ini()
{
    ms( add , -1 );
    rep( i , n ) deg[i] = Isshop[i] = 0 ;
}
pii DP( int now , int mask )
{
    int &v = vis[now][mask];
    pii &ret = dp[now][mask];
    if( v == cs ) return ret ;
    v = cs ;
    ret = mp( -INF , 0 );
    if( add[now][s+1] != -1 ) ret = mp( 0 , add[now][s+1]);
    pii tmp ;
    //printf("now:: %d\n",now);
    rep( i , s + 1 )
    {
        if( mask & ( 1 << i ) ) continue ;
        if( add[now][i] == -1 ) continue ;
        tmp = DP( i , mask | ( 1 << i ) );
        tmp.ff++;
        tmp.ss += add[now][i];
        if( tmp.ff > ret.ff ) ret = tmp ;
        else if( tmp.ff == ret.ff && tmp.ss < ret.ss ) ret = tmp ;
        //cout << " now " << ret.ff << " " << ret.ss << endl ;
    }
    return ret ;
}
void Dijkstra( int now , int idx )
{
    rep( i , n )
    {
        dis[i] = -1 ;
        mem[i] = 0 ;
    }
    priority_queue < pii > pq ;
```

```

dis[now] = 0 ;
pq.push(mp(0,now));
while( !pq.empty())
{
    pii prv = pq.top();
    pq.pop();
    int pc = (-1)*(prv.ff);
    int x = prv.ss ;
    if( mem[x] ) continue ;
    mem[x] = 1 ;
    rep( i , deg[x])
    {
        int u = adj[x][i];
        int nc = pc + cost[x][i];
        if( mem[u] ) continue ;
        if( dis[u] == -1 || dis[u] > nc )
        {
            dis[u] = nc ;
            pq.push(mp(-nc,u));
        }
    }
}
rep( i , s + 1)
{
    add[idx][i] = dis[shop[i]];
}
add[idx][s+1] = dis[n-1];
}

int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // ms( vis , -1 );
    // cout << fixed << setprecision(10);
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II , m = II , s = II;
        ini();
        int scr = 0 , sink = 0 ;
        For( i , s ) shop[i] = II ;
        rep( i , m )
        {
            int x = II , y = II , c = II ;
            adj[x][deg[x]] = y ;
            cost[x][deg[x]++] = c ;

```

```
    }  
    rep( i , s + 1 ) Dijkstra(shop[i],i);  
    pii ans = DP(0,1);  
    printf("Case %d: ",cs);  
    if( ans.ff < 0 ) puts("Impossible");  
    else printf("%d %d\n",ans.ff,ans.ss);  
    }  
    return 0;  
}
```

1326 - Race

Time Limit: 1 second(s)

Memory Limit: 32 MB

Disky and Sooma, two of the biggest mega minds of Bangladesh went to a far country. They ate, coded and wandered around, even in their holidays. They passed several months in this way. But everything has an end. A holy person, Munsiji came into their life. Munsiji took them to derby (horse racing). Munsiji enjoyed the race, but as usual Disky and Sooma did their as usual task instead of passing some romantic moments. They were thinking- in how many ways a race can finish! Who knows, maybe this is their romance!

In a race there are n horses. You have to output the number of ways the race can finish. Note that, more than one horse may get the same position. For example, 2 horses can finish in 3 ways.

1. Both first
2. horse1 first and horse2 second
3. horse2 first and horse1 second

Input

Input starts with an integer T (≤ 1000), denoting the number of test cases.

Each case starts with a line containing an integer n ($1 \leq n \leq 1000$).

Output

For each case, print the case number and the number of ways the race can finish. The result can be very large, print the result modulo 10056.

Sample Input	Output for Sample Input
3	Case 1: 1
1	Case 2: 3
2	Case 3: 13
3	

SOLUTION

```
#include <iostream>
#include <stdio.h>
#define N 1003
#define MOD 10056
using namespace std;
long long dp[N+5];
long long count[N + 5];
int main()
{
    int n;
    int t;

    long long sum;

    count[1] = 1;
    dp[1] = 1;
    for (int i = 2; i <= N; i++) {
        sum = 1;
        for (int j = i; j >= 2; j--) {
            count[j] = (j * count[j]) % MOD + (j * count[j - 1])
% MOD;

            sum = (count[j] + sum) % MOD;
        }
        dp[i] = sum;
        dp[i] = dp[i] % MOD;
    }

    scanf("%d", &t);
    for (int cs = 1; cs <= t; cs++) {
        scanf("%d", &n);
        printf("Case %d: %d\n", cs, dp[n]);
    }
}
```


1327 - Help the Winners

Time Limit: 2 second(s)

Memory Limit: 32 MB

The Sell N' Profit shop has recently had a raffle contest with a prize of a dress and matching shoes to some of their regular customers. They have set aside an equal number of dresses and pairs of shoes of different designs for the purpose.

The plan was going quite well until one customer pointed out that not all the dresses and all the pairs of shoes match each other (i.e. you won't look good if you wore that dress and that pair of shoes at the same time - but the same dress or shoes might look good with something else). So, simply giving someone a dress and a pair of shoes at random, risks angering their customers. They have to give each winner a dress and shoes that match. But some pairs of (dress, shoes) are so adorable to girls such that if they are given to any girl, she would be happier than ever and she may become a great customer of the shop. These pairs are called super matching pairs. But, choosing such pair may lead to a situation that some other girls get non-matching (dress, shoes) pairs.

Now you are given n dresses and n pairs of shoes, a list of what dress matches which shoes and some super matching (dress, shoes) pairs. As the company have lack of fashion sense, they ask you to find how many ways there are to make up n sets of dresses and matching shoes, such that either all the girls are happy (got matching dress and shoes), or at least one of the girls got a super matching pair (in this case some girls may get non matching pairs).

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with an integer n ($1 \leq n \leq 15$) by itself on a line, denoting the number of dresses and pair of shoes. Each of the next n lines contains n space separated integers (one of 0, 1 or 2). The j^{th} integer in i^{th} line represents the status of the i^{th} dress and j^{th} pair of shoes. 0 means that i^{th} dress doesn't match j^{th} pair of shoes. 1 means they match. 2 means they are super matching.

Output

For each case, print the case number and number of ways you can form n sets of dresses and shoes.

Sample Input	Output for Sample Input
2 3 0 1 1 1 1 0 1 0 1 3	Case 1: 2 Case 2: 4

1	1	2	
2	1	0	
1	1	2	

SOLUTION

```
const int NX = 1000 ;
const int mod = 10056 ;
Long dp[ ( 1 << 16 ) ][2][2] ;
int vis[ ( 1 << 16 ) ][2][2] ;
int inp[18][18] , n , cs;
Long DP( int mask , int row ,int sup , int all )
{
    if( mask == ( 1 << n ) - 1 ) return ( sup || all );
    int &v = vis[mask][sup][all];
    Long &ret = dp[mask][sup][all];
    if( v == cs ) return ret ;
    ret = 0 ;
    v = cs ;
    int i ;
    rep( i , n )
    {
        if( mask & ( 1 << i ) ) continue ;
        ret += DP( mask | ( 1 << i ) , row + 1 , sup || inp[row][i]==2 , all &&
inp[row][i] == 1 );
    }
    return ret ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // cout << fixed << setprecision(10) ;
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II ;
        rep( i , n ) rep( j , n ) inp[i][j] = II ;
        printf("Case %d: %lld\n",cs,DP(0,0,0,1));
    }
    return 0;
}
```

1329 - Playing Cards

Time Limit: 1 second(s)

Memory Limit: 32 MB

In a regular card set there are 52 cards, each card has two parts, the value and the suit. The values are one of 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A and the suit is one of H, S, D, C. Cards are represented first by their value and then by their suit. For example, 7H, 2C, JD etc are some cards.

Now from the regular card set, you removed some of the cards (probably 0). Your task is to find the number of ways you can place the cards in a line such that no two adjacent cards have the same value. You have to use all the cards (of course the remaining cards).

For example, 2H, 3C, 5C is a valid solution, but 5H, 5C, 7S is not.

Input

Input starts with an integer **T** (≤ 20000), denoting the number of test cases.

Each case starts with a line containing an integer **n** ($1 \leq n \leq 52$) denoting the number of **remaining** cards, followed by **n** remaining cards (a single space precedes every card). Assume that the cards are from the regular set and no card is reported more than once. Also assume that the cards will be represented as stated above.

Output

For each case, print the case number and the number of ways to place the cards in a line such that no two adjacent cards have the same value. Print the result modulo 2^{64} .

Sample Input	Output for Sample Input
5	Case 1: 1
1 TC	Case 2: 0
2 TC TS	Case 3: 48
5 2C AD AC JC JH	Case 4: 24
4 AC KC QC JC	Case 5: 120
6 AC AD AS JC JD KD	

SOLUTION

```
ull dp[15][15][15][15][5] ;
int vis[15][15][15][15][5] , cs , n ;
char str[10];
int A[5] , freq[15] ;
int chk( char c )
{
    if( c == 'T' ) return 10 ;
    if( c == 'J' ) return 11 ;
    if( c == 'Q' ) return 12 ;
    if( c == 'K' ) return 13 ;
    if( c == 'A' ) return 1 ;
    return (c - '0');
}
ull DP( int one , int two , int three , int four , int lst )
{
    // printf(" %d %d %d %d %d\n",one,two,three,four,lst);
    if( !one && !two && !three && !four ) return 1 ;
    ull &ret = dp[one][two][three][four][lst];
    int &v = vis[one][two][three][four][lst];
    if( v ) return ret ;
    v = 1 ;
    ret = 0 ;
    if( one )
    {
        // printf("one\n");
        if( lst == 2 ) ret += ( one - 1 ) * DP( one - 1 , two ,
three , four , 1 );
        else ret += ( one ) * DP( one - 1 , two , three , four
, 1 );
    }
    if( two )
    {
        if( lst == 3 ) ret += (ull) 2 * ( two - 1 ) * DP( one +
1 , two - 1 , three , four , 2);
        else ret += (ull) 2 * ( two ) * DP( one + 1 , two - 1 ,
three , four , 2 );
    }
    if( three )
    {
        if( lst == 4 ) ret += (ull) 3 * ( three - 1 ) * DP( one
, two + 1 , three - 1 , four , 3 );
        else ret += (ull) 3 * ( three ) * DP( one , two + 1 ,
three - 1 , four , 3 );
    }
}
```

```

    }
    if(four ) ret += (ull) 4 * ( four ) * DP( one , two , three
+ 1 , four - 1 , 4 );
    return ret ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good
human being
    // ms( vis , -1 );
    // cout << fixed << setprecision(10);
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        n = II ;
        ms( A , 0 );
        ms( freq , 0 );
        rep( i , n )
        {
            scanf("%s",str);
            freq[chk(str[0])]++;
        }
        for( int i = 1 ; i <= 13 ; i++ ) A[freq[i]]++;
        printf("Case %d:
%llu\n",cs,DP(A[1],A[2],A[3],A[4],0));
    }
    return 0;
}

```

1334 - Genes in DNA

Time Limit: 2 second(s)

Memory Limit: 32 MB

Your friend is a biologist. He has just sequenced a DNA and wants to know about contribution of different genes in that DNA. Both Gene and DNA can be represented by a sequence of letters or strings. Given the sequence of a DNA **D** and a Gene **G**; your friend uses following method to calculate the contribution.

1. Generate a list **P** of proper non-empty prefixes of **G** and another **S** of proper non-empty suffixes of **G** [1]. Additionally let the **L** is list of all strings that is concatenation of a prefix and a suffix. So if **G** = **ACCT** then **P** = **A**, **AC**, **ACC** and **S** = **T**, **CT**, **CCT** and **L** = **AT**, **ACT**, **ACCT**, **ACT**, **ACCT**, **ACCCT**, **ACCT**, **ACCCT**, **ACCCCT**. If $|G| = n$ then it is obvious that size of **L** is $(n - 1)^2$.
2. For each element of **L**, count number of times it occurs as substring in **D**. Contribution of Gene **G** in DNA **D** is total of these values. For example if **D** = **ACTACCTACCCCT** then

AT	0
ACT	1
ACCT	1
ACT	1
ACCT	1
ACCCT	0
ACCT	1
ACCCT	0
ACCCCT	1
Total	6

As this process is very clumsy he wants to automate this process. As he is not a programmer, he needs your help. He will be very grateful if you kindly write him a program which will read the sequence of the DNA and the Gene, and will calculate contribution of the Gene in the DNA.

Input

Input starts with an integer **T** (≤ 20), denoting the number of test cases.

Each case contains two lines. The first line contains a string denoting the sequence of DNA, and the second line contains another string denoting the Gene. The length of each string is less than **50000** and consists of only **A, C, T** and **G**.

Output

For each case, print the case number and the contribution, as described above.

Sample Input	Output for Sample Input
3 ACTACCTACCCCT ACCT AAA AAAA AAAA AAA	Case 1: 6 Case 2: 4 Case 3: 8

Note

1. Proper prefix (suffix) of a string **S** is a prefix (suffix) of length smaller than **|S|**. Here **|S|** denotes length of **S**.
2. Dataset is huge, use faster I/O methods.

SOLUTION

```
const int NX = 50000 + 10 ;
char T[NX] , P[NX] ;
Long dp[NX] , matchLeft[NX] , matchRight[NX] ;
int pi[NX] , f[NX] ;
void KmpMatch(char *t , char *p , Long *m)
{
    int tt = strlen(t);
    int pp = strlen(p);
    int k = 0 , i = 0 ;
    pi[0] = pi[1] = 0 ;
    if( pp > 1 ) dp[1] = 1 ;
    //failure-func
    for( i = 2 ; i < pp ; i++ )
    {
        while( k > 0 && p[i-1] != p[k] ) k = pi[k];
        if( p[i-1] == p[k] ) k++;
        pi[i] = k ;
        dp[i] = dp[k] + 1 ;
    }
    k = 0 ;
    //pattern look-up
    for ( i = 0 ; i < tt ; i++ )
    {
        while( k > 0 && t[i] != p[k] ) k = pi[k];
        if( p[k] == t[i] ) k++;
        m[i] = dp[k];
        if( k == pp - 1 ) k = pi[k];
    }
}

int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    // ms( vis , -1 );
    // cout << fixed << setprecision(10);
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        scanf("%s %s",T,P);
        int tt = strlen(T);
        int pp = strlen(P);
        KmpMatch(T,P,matchLeft);
        reverse(T,T+tt);
```

```

        reverse(P,P+pp);
        KmpMatch(T,P,matchRight);
        reverse(matchRight , matchRight + tt );
        Long ans = 0 ;
        for ( int i = 0 ; i < tt - 1; i++ ) ans += (111 * matchLeft[i] *
matchRight[i+1]);
        printf("Case %d: %lld\n",cs,ans);
    }
    return 0;
}

```

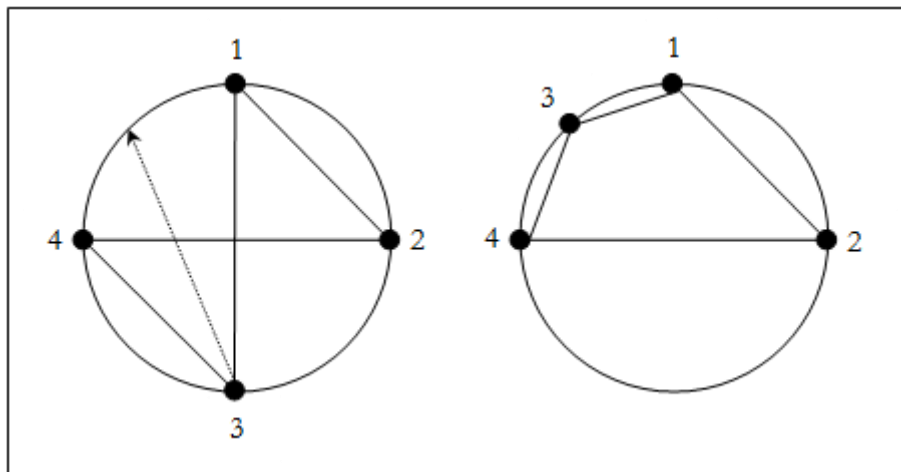
1335 - Planar Graph

Time Limit: 4 second(s)

Memory Limit: 32 MB

A graph is said to be a **planar graph** if it can be drawn on a plane such a way that its edges intersect only at their end points. In other words, it can be drawn in such a way that no edges cross each other.

In this problem, you are given a 2-regular connected graph where the n vertices (numbered from 1 to n) are drawn in the perimeter of a circle in clockwise order. Now the graph may or may not be planar. We want to make it planar. We are allowed to move a vertex to any empty place in the perimeter of the circle. When a vertex is moved, its edges will also move with it, but remain connected with the vertex.



For example, there is a non-planar graph in the left picture, we made it a planar graph (shown in the right picture) by moving vertex-3 between vertex 1 and 4. There are other solutions, but the best we can do is to move one vertex and make it planar. You have to do the similar task, you are given a graph as described, and your task is to make it planar by moving as few vertices as possible. Assume that the circle is large enough such that you can place any number of vertices between two particular vertices.

Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each case starts with a line containing an integer n ($3 \leq n \leq 500$). Each of the next n lines contains two integers. The i^{th} line ($1 \leq i \leq n$) contains the vertices that have an edge with vertex i . You can assume that the given graph follows the restrictions defined above.

Output

For each case, print the case number first. Then if it's impossible to make it planar, print '**impossible**' in the same line. Otherwise, print the minimum possible number of vertices that need to be moved to make the graph planar.

Sample Input	Output for Sample Input
2 4 2 3 4 1 4 1 3 2 6 5 4 5 3 2 6 1 2 1 4 3 6	Case 1: 1 Case 2: 2

Note

1. A graph is said to be **2-regular** if each vertex has exactly two edges.
2. A **connected** graph is an undirected graph where it's possible to move from any vertex to another using the edges.

SOLUTION

```
#include <stdio>
#include <cstring>
#include <vector>
#include <algorithm>
using namespace std;
int G[512][2], ord[512];
char vis[512];
void dfs(int u, int d) {
    ord[u] = d; vis[u] = 1;
    for(int i = 0; i < 2; i++) {
        if(!vis[G[u][i]]) dfs(G[u][i], d + 1);
    }
}
int lis(int n) {
    vector< int > V;
    V.push_back(ord[1]);
    for(int i = 2; i <= n; i++) {
        if(ord[i] > V.back()) V.push_back(ord[i]);
        else *lower_bound(V.begin(), V.end(), ord[i]) = ord[i];
    }
    return V.size();
}
int main() {
    int test, cs, n, i, res;
    //freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
    scanf("%d", &test);
    for(cs = 1; cs <= test; cs++) {
        scanf("%d", &n);
        for(i = 1; i <= n; i++) scanf("%d %d", &G[i][0], &G[i][1]);
        for(res = n, i = 1; i <= n; i++) {
            memset(vis, 0, sizeof vis);
            dfs(i, 0);
            res = min(res, n - lis(n));
            swap(G[i][0], G[i][1]);
            memset(vis, 0, sizeof vis);
            dfs(i, 0);
            res = min(res, n - lis(n));
        }
        printf("Case %d: %d\n", cs, res);
    }
    return 0;
}
```


1342 - Aladdin and the Magical Sticks

Time Limit: 3 second(s)

Memory Limit: 32 MB

It's said that Aladdin had to solve seven mysteries before getting the Magical Lamp which summons a powerful Genie. Here we are concerned about the second mystery.

Aladdin was going along a magical cave, tricked by the evil sorcerer, searching for the great Magical Lamp. Then he found a strange Lamp, thinking that it might be the lamp the sorcerer had spoken off, rubbed it and a Genie appeared. But alas! He was unlucky as the lamp was owned by an evil Genie. It blindfolded Aladdin and gave him a task to finish. Aladdin finished that task and moved forward.

The task was that, there were n sticks, each had a particular weight. Two types of sticks were there, one kind of sticks had distinguishable rough patterns that can be identified by just touching. Other types of sticks were indistinguishable. Each time Aladdin had to pick a stick from all sticks and put it into a magical box. If all the sticks are put into the box **at least once**, Aladdin will be free; otherwise the stick he just put will be put with the other sticks.

So, Aladdin planned that each time he will pick a **new** stick randomly and put it into the magical box. So, once he put a distinguishable stick into the box, he will not put it again though it's mixed with other sticks, since he can remember the roughness of every distinguishable stick. But for indistinguishable sticks he had no option. So, each time he put a stick that looked new to him, but that might not be a new one. And each time the probability of picking a new (to Aladdin of course!) stick is equal. Now your task is to find the **expected** summation of weights of sticks Aladdin had to put into the box before he was free.

For example, let's say there were two sticks, one was distinguishable and the other one was indistinguishable. And the weights were 4, 8 respectively. Let's calculate the expected summation of weights of sticks.

- 1) Aladdin puts stick 1 (weight 4). The probability is $1/2$. As all the sticks are **not** put into the box at least once, so, it will be put with the other stick again. Aladdin will not put stick 1 again, since he can distinguish it from others. So, he puts stick 2 next. And since all the sticks are put in the box at least once, so he is free. Total weight he put is $4 + 8 = 12$. So, the expected summation of weights is $(1/2) * 12 = 6$.
- 2) Let's say he puts stick 2 (weight 8). The probability is $1/2$. Now it will be put back with the other stick. So, Aladdin has two sticks again, where he is not sure which one he had put into the box, as he cannot distinguish stick 2. So, he kept trying stick 1 and 2. And if he puts stick 2 again, he faces the same situation, but if he puts stick 1, he is free as all the sticks are put into the box at least once. Let's say the expected summation of weights of sticks from this situation is x . If he puts stick 1 (weight 4, but probability is $1/2$), he is free. If he puts

stick 2 (weight 8), he is in the same situation. So, $x = (1/2) * 4 + (1/2) * (x + 8)$. So, $x = 12$. So, the expected summation of weights is $(1/2) * (8 + 12) = 10$.

So, from the both outcomes, the final result is $6 + 10 = 16$.

Input

Input starts with an integer **T** (≤ 10), denoting the number of test cases.

Each case starts with a line containing an integer **n** ($1 \leq n \leq 5000$). Each of the next **n** lines contains two integers **a_i b_i** ($1 \leq a_i \leq 5000$, $1 \leq b_i \leq 2$) where **a_i** denotes the length of the stick and **b_i** denotes the type. **b_i = 1** indicates that the stick is distinguishable from others, **b_i = 2** means it is not distinguishable.

Output

For each case, print the case number and the expected summation of weights of the sticks Aladdin had to put into the box. Errors less than 10^{-4} will be ignored.

Sample Input	Output for Sample Input
4	Case 1: 16
2	Case 2: 11
4 1	Case 3: 10.5000000000
8 2	Case 4: 13.8333333333
2	
5 1	
6 1	
2	
2 2	
5 2	
3	
1 1	
2 2	
5 2	

SOLUTION

```
using namespace std;
typedef long long ll;
typedef unsigned long long llu;
int n, k, ks;
vector<int> sticks[2];
//double dp[5001][5001];
double tot[2], avgIndist, Hn[5001];
//int vis[5001][5001];
/* p = currently untouched, q = currently untouched distinguishable
   p - q currently untouched indistinguishable
   n - p currently touched, k - q currently touched distinguishable
   n + q - p - k currently touched indistinguishable
   currently n - k + q sticks onboard
   (n - k + q)*E(p,q) = (E(p-1, q) + sid)*(p-q) + (E(p, q) + sid)*(n+q-
p-k) + (E(p-1,q-1)+sd)*q
   p*E(p,q) = E(p-1,q)*(p-q) + (E(p-1,q-1 + sd)*(q) + totalIndist
*/
double calc(int p, int q){
    double tmp[5001], dp1[5001];

    for ( int i = 0 ; i <= p ; i++ ){
        dp1[i] = tmp[i] = 0;
    }

    for ( int i = 1 ; i <= p ; i++ ){
        for ( int j = 0 ; j <= q ; j++ ) tmp[j] = dp1[j];
        for ( int j = 0 ; j <= q ; j++ ){
            if ( i - j > p - q || j > i ){
                dp1[j] = 0;
                continue;
            }
            dp1[j] = tot[1] + (i - j) * tmp[j];
            if ( j ) dp1[j] += j * ( tmp[j-1] + tot[0] / k );
            dp1[j] /= i;
        }
    }

    return dp1[q];
}

/*double calc(int p, int q){
    return Hn[p] * tot[1] + tot[0];
}
*/
```

```

int main(){
    int test, u, v;

    Hn[0] = 0;
    for ( int i = 1 ; i <= 5000 ; i++ ) Hn[i] = Hn[i-1] + 1.0/i;
    while ( scanf("%d", &test) == 1 ){
        for ( ks = 1 ; ks <= test ; ks++ ){
            scanf("%d", &n);
            k = 0;
            tot[0] = tot[1] = 0; // tot[i], total weight of sticks
of type i

            sticks[0].clear();
            sticks[1].clear();
            for ( int i = 0 ; i < n ; i++ ){
                scanf("%d%d", &u, &v); // u = weight, v = type,
v == 1, the stick is distinguishable by touch, v==2 it isn't
                v--;
                sticks[v].pb(u);
                if ( !v ) k++;
                tot[v] += u;
            }

            printf("Case %d: %.10lf\n", ks, calc(n, k));
        }
    }

    return 0;
}

```

1344 - Aladdin and the Game of Bracelets

Time Limit: 2 second(s)

Memory Limit: 32 MB

It's said that Aladdin had to solve seven mysteries before getting the Magical Lamp which summons a powerful Genie. Here we are concerned about the fourth mystery.

In the cave, Aladdin was moving forward after passing the pathway of magical stones. He found a door and he was about to open the door, but suddenly a Genie appeared and he addressed himself as the guardian of the door. He challenged Aladdin to play a game and promised that he would let Aladdin go through the door, if Aladdin can defeat the Genie. Aladdin defeated the Genie and continued his journey. However, let's concentrate on the game.

The game was called 'Game of Bracelets'. A bracelet is a linear chain of some pearls of various weights. The rules of the game are:

- 1) There are n bracelets.
- 2) Players alternate turns.
- 3) In each turn, a player has to choose a pearl from any bracelet. Then all the pearls from that bracelet, that were **not lighter** than the pearl, will be removed. It may create some smaller bracelets or the bracelet will be removed if no pearl is left in the bracelet.
- 4) The player, who cannot take a pearl in his turn, loses.

For example, two bracelets are: 5-1-7-2-4-5-3 and 2-1-5-3, here the integers denote the weights of the pearls. Suppose a player has chosen the first pearl (weight 5) from the first bracelet. Then all the pearls that are not lighter than 5, will be removed (from first bracelet). So, ~~5~~-1-~~7~~-2-4-~~5~~-3, the red ones will be removed and thus from this bracelet, three new bracelets will be formed, 1, 2-4 and 3. So, in the next turn the other player will have four bracelets: 1, 2-4, 3 and 2-1-5-3. Now if a player chooses the only pearl (weight 3) from the third bracelet, then the bracelet will be removed.

Now you are given the information of the bracelets. Assume that Aladdin plays first, and Aladdin and the Genie both play optimally. Your task is to find the winner.

Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each case starts with a line containing an integer n ($1 \leq n \leq 50$). Each of the next n lines contains an integer K_i ($1 \leq K_i \leq 50$) followed by K_i integers, denoting the weights of the pearls of the i^{th} ($1 \leq i \leq n$) bracelet. Each weight will lie in the range $[1, 10^5]$.

Output

For each case, print the case number and '**Aladdin**' if Aladdin wins. Otherwise print '**Genie**'. If Aladdin wins, then print an additional line, denoting the weight of pearls, one of which should be chosen in the **first** move by Aladdin such that he can win. Each pearl should be denoted by a pair of integers: the first integer is the bracelet number and the second integer is the weight. Sort the pearls first by the bracelet number then by the weight. And same weight in a bracelet should be reported once. Check the samples for exact formatting.

Sample Input	Output for Sample Input
4 2 7 5 1 7 2 4 5 3 4 2 1 5 4 2 2 5 2 2 5 2 1 5 5 2 5 2 5 3 5 5 2 5 2 5 5 7 2 7 3 2 4 5 1 5 4	Case 1: Aladdin (2 5) Case 2: Genie Case 3: Aladdin (1 2) (1 5) Case 4: Aladdin (2 7) (3 1) (3 5)

SOLUTION

Can't find any!

1345 - Aladdin and the Happy Garden

Time Limit: 2 second(s)

Memory Limit: 64 MB

It's said that Aladdin had to solve seven mysteries before getting the Magical Lamp which summons a powerful Genie. Here we are concerned about the fifth mystery.

After defeating the Genie in 'Game of Bracelets', Aladdin moved forward and found a garden. There were n small Genies in the garden and they were all sad. Aladdin asked them about the lamp. But none of them was interested. So, Aladdin had to make them happy. Soon he noticed that height of every Genie is distinct but they were standing in a line randomly. Aladdin found a book there, and after reading the book, he discovered that the only way to make them happy was to make a line with the Genies such that k consecutive genies in the line could be found whose heights are in increasing order. But the book also mentioned that if $k+1$ consecutive Genies found in the line whose heights are in increasing order then they would be sad again. So, Aladdin did make them happy, and they showed the path to Aladdin and he moved forward.

Now you are given n and k , your task is to find in how many ways Aladdin could make them happy.

Input

Input starts with an integer T (≤ 1275), denoting the number of test cases.

Each case starts with a line containing two integers: n and k ($1 \leq k \leq n \leq 50$).

Output

For each case, print the case number and the number of ways Aladdin could make them happy. As the result can be big; print the result modulo 1000 000 007.

Sample Input	Output for Sample Input
5	Case 1: 1
2 2	Case 2: 4
3 2	Case 3: 6
4 3	Case 4: 8
5 4	Case 5: 41
5 3	

SOLUTION

```
const int NX = 52 ;
const Long mod = 1000000007;
Long dp[NX][NX][NX][2];
int k , n , vis[NX][NX][NX][2] , tt ;
Long ans[NX][NX];
Long DP( int Small , int Big , int Cur , int found )
{
    if( Cur == k ) found = 1 ;
    if( Cur > k ) return 0 ;
    if( Small + Big == 0 ) return (found );
    int &v = vis[Small][Big][Cur][found];
    Long &ret = dp[Small][Big][Cur][found];
    if( v == tt ) return ret ;
    v = tt ;
    ret = 0 ;
    int i ;
    // Placing Small number
    for ( i = 1 ; i <= Small ; i++ ) { ret += DP( i - 1 , Small + Big - i , 1 ,
found );
        ret %= mod ; }
    // placing Big numbe
    if( Cur != k )
        for( i = 1 ; i <= Big ; i++ ) {
            ret += DP( Small + i - 1 , Big - i , Cur + 1 , found );
            ret %= mod ;
        }
    return ret ;
}

void pre()
{
    for( k = 1 , tt = 1 ; k <= 50 ; k++ , tt++ )
    {
        for( n = k ; n <= 50 ; n++ )
        {
            ans[n][k] = DP( 0 , n , 0 , 0 );
        }
    }
}

int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    pre();
```

```
int cs , t = II ;
for ( cs = 1 ; cs <= t ; cs++ )
{
    n = II , k = II ;
    printf("Case %d: %lld\n",cs,ans[n][k]);
}
return 0;
}
```


1351 - Ordered Flips

Time Limit: 2 second(s)

Memory Limit: 32 MB

You are given two non-empty strings **X** and **Y** of same length **n**. Your task is to make them identical. But the problem is that the only operation you can do is flipping, and it can only be applied to **X**.

For a flip, two positions of the string **X** are chosen, let the positions be **i** and **j** ($0 \leq i < j < n$) and if you apply **flip (i, j)** all characters between **i** and **j** (inclusive) are reversed. For example, let **X** be "abcdefg", then if you apply flip (2, 5) to **X** then **X** will be "ab**fedc**g". But if you want to apply flips more than once, you have to use ordered flips. If **flip (i₂, j₂)** is applied immediately after **flip (i₁, j₁)**, then it will be said "**Ordered Flips**" if and only if $i_1 \leq i_2$ and $j_2 \leq j_1$.

So, now your task is to find the minimum number of ordered flips to change **X** to **Y**.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case contains two lines, each containing a non empty string of length **n** ($1 \leq n \leq 60$). The strings contain lowercase English letters only. First line contains **X** and second line contains **Y**.

Output

For each case, print the case number and the minimum number of ordered flips needed to change **X** to **Y**. If it's impossible to do, then print "**impossible**". Check the samples for details.

Sample Input	Output for Sample Input
4 abcd dcba abca aabc zzzaaazzzaaa aazzaazzaazz aab bab	Case 1: 1 Case 2: 2 Case 3: 4 Case 4: impossible

SOLUTION

```
const int NX = 62 ;
const int INF = 1 << 28 ;
int dp[2][2][NX][NX][NX] , vis[2][2][NX][NX][NX] , cs ;
char X[NX] , Y[NX];
int DP( int flip_allowed , int flipped , int i , int j , int k)
{
    if( i >= j ) return 0 ;
    int &v = vis[flip_allowed][flipped][i][j][k];
    int &ret = dp[flip_allowed][flipped][i][j][k];
    if( v == cs ) return ret ;
    v = cs ;
    ret = INF ;
    if( flip_allowed ) ret = min( ret , 1 + DP(0,!flipped,i , j , k ) );
    if( !flipped )
    {
        if( X[i] == Y[k] ) ret = min( ret , DP( 1 , 0 , i + 1 , j , k + 1 ) );
        if( X[j] == Y[k+j-i] ) ret = min( ret , DP( 1 , 0 , i , j - 1 , k ) );
    }
    else
    {
        if( X[j] == Y[k] ) ret = min( ret , DP( 1 , 1 , i , j - 1 , k + 1 ) );
        if( X[i] == Y[k+j-i] ) ret = min( ret , DP( 1 , 1 , i + 1 , j , k ) );
    }
    return ret ;
}

int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human being
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        scanf("%s%s",X,Y);
        int len = strlen(X);
        if( strcmp(X,Y) == 0 )
        {
            printf("Case %d: 0\n",cs);
            continue ;
        }
        int ans = DP( 1 , 0 , 0 , len - 1 , 0 );
        if( ans == 0 || ans >= INF ) printf("Case %d: impossible\n",cs);
        else printf("Case %d: %d\n",cs,ans);
    }
}
```

```
    return 0;  
}
```

1352 - Strange Summation

Time Limit: 2 second(s)

Memory Limit: 32 MB

Mr Foolizm is learning binary numbers. He has a software that converts any decimal integer to binary and vice versa. So, if he is asked to add some decimal numbers, he first converts them to binary. Then he adds them in binary. But he doesn't understand the carry principle yet and he even doesn't know the idea of positioning the numbers. Instead of LSD (least significant digit), he starts summing from MSD (most significant digit). So, if he is asked to add four integers from 1 to 4, he does the following:

1. He first takes 1 and 2 and convert them to binary using his software, and then he adds them like the following:

```
1
10
----
00
```

2. Then he takes 3, converts it to binary and add it with the previous result

```
00
11
----
11
```

3. He then takes 4, converts it to binary and add it with the previous result

```
11
100
----
010
```

4. And finally he converts the result back to decimal, so, the result is 2.

Now you are given two integers **p** and **q**, your task is to find the result if Mr Foolizm adds all integers from **p** to **q** (inclusive) in his procedure.

Input

Input starts with an integer **T** (≤ 10000), denoting the number of test cases.

Each case starts with a line containing two integers: **p** and **q** ($0 < p < q < 2^{63}$).

Output

For each case, print the case number and the result.

Sample Input	Output for Sample Input
4	Case 1: 2
1 4	Case 2: 3
2 5	Case 3: 1610612736
1 2147483647	Case 4: 2
7 12	

SOLUTION

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<algorithm>
#include<cmath>
#include<vector>
#include<queue>
#include<map>
#include<set>
#include<ctime>
using namespace std;
typedef long long ll;
#define INF 0x3f3f3f3f
#define maxn 111
ll cnt[maxn];
void Solve(ll n,int sta)
{
    cnt[0]+=n*sta;
    for(int i=1;i<63;i++)
    {
        ll m=111<<i,len=1;
        if(m>n)break;
        while(m<=n/2)
        {
            cnt[i]+=m/2*sta;
            m<=<111,len<=<111;
        }
        m=n-m+1;
        cnt[i]+=(m/len/2*len)*sta;
        if((m/len)&1)cnt[i]+=(m%len)*sta;
    }
}
int main()
{
    int T,Case=1;
    ll l,r;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%lld%lld",&l,&r);
        memset(cnt,0,sizeof(cnt));
        Solve(r,1),Solve(l-1,-1);
        int res=0;
        while(r)res++,r>>=1;
        ll ans=0;
        for(int i=0;i<res;i++)
            if(cnt[i]&1)ans+=111<<(res-1-i);
        printf("Case %d: %lld\n",Case++,ans);
    }
    return 0;
}
```


1360 - Skyscraper

Time Limit: 2 second(s)

Memory Limit: 32 MB

The Build n' Profit construction company is about to build its tallest building. It will be huge, the tallest building in the world by a wide margin. It will house hundreds of thousands of people and have rocket-powered elevators to the upper floors. They even plan for a shuttle docking station instead of a helipad on its roof. But any building of that size is extremely costly to build and maintain, and even after selling and renting out all the floor-space it will be very difficult to meet the costs. Luckily, they have come up with a great solution. They will place advertisements on the outer walls of the building for a hefty charge. This will help offset some of the costs and bring in a profit.

However, feedbacks from prospective buyers of this advertisement space have brought up a new problem. Each customer wants a specific sized advertisement placed at a specific height, and they will pay a certain amount of money for it. Each advertisement order specifies its position (i.e. the lowest floor of the advertisement) and its size (i.e. the number of floors it covers, including the starting floor). Each advertisement spans the whole face of the building, so no two advertisements can occupy the same floor and no floors can be partially covered. Each order also includes the amount to be earned if that advertisement is placed on the building. Of course, no money is earned if only part of an advertisement is placed, or it is placed in any other position.

Since many of the advertisements want some of the same floors as others, it is often impossible to choose all of them. Can you help choosing which of the orders to accept so that the above constraints are fulfilled and the amount of profit is maximized?

Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each case starts with an integer N ($1 \leq N \leq 30000$) denoting the number of advertisement orders. Each of the next N lines represents an advertisement by three integers A ($0 \leq A \leq 10^5$), B ($1 \leq B \leq 10^5$) and C ($1 \leq C \leq 1000$) denoting the lowest floor, the number of floors the advertisement covers (including the lowest floor) and the amount of money earned for placing it, respectively.

Output

For each case, print the case number and the maximum profit they can achieve.

Sample Input	Output for Sample Input
1 3 1 5 1	Case 1: 3

2 10 3	
7 12 1	

Note

Dataset is huge, use faster I/O methods.

SOLUTION

```
const int MAXN = 30000+5 ;
struct abc
{
    int start , end , cost;
} inp [ MAXN ] ;
int Indxing [ MAXN ] ;
LL dp[ MAXN ];
bool cmp ( abc A , abc B )
{
    if(A.end == B.end ) return A.start < B.start ;
    return A.end < B.end ;
}
LL solve(int n )
{
    sort ( inp+1, inp+n+1 , cmp );
    sort (Indxing+1,Indxing+n+1);
    int i;
    memo(dp,0);
    inp[0].start = inp[0].end = -1;
    Indxing[0] = -1;
    LL ans = -1;
    for ( i = 1 ; i <= n ; i++ )
    {
        int idx = lower_bound (Indxing , Indxing+i , inp[i].start ) -
Indxing ;
        if( idx && Indxing[idx] >= inp[i].start ) idx--;
        dp[i] = dp[idx] + inp[i].cost ;
        dp[i] = max ( dp[i] , dp[i-1] );
        ans = max ( ans , dp[i] );
    }
    return ans ;
}
int main()
{
    int cs, t , i , n;
    scanf("%d",&t);
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        scanf("%d",&n);
        for ( i = 1; i <= n ; i++ )
        {
            scanf("%d %d %d",&inp[i].start , &inp[i].end ,
&inp[i].cost ) ;
            inp[i].end += inp[i].start - 1 ;
        }
    }
}
```

```
        Indxing[i] = inp[i].end ;
    }
    printf("Case %d: %lld\n",cs,solve(n));
}
}
```

1362 - Electricity Connection

Time Limit: 6 second(s)

Memory Limit: 32 MB

The city 'AjobakahD' has a lot of problems with electricity. Load shedding is a common problem here and people are quite used to it. Instead of calculating the total time the power was on, they calculate the total time the power was off. And of course the later one is always greater.

There is a small area in the city which has not yet been enlightened with any load shedding! That means they haven't got the electricity connection yet. Now the Power Development Board (PDB) wants to set electricity connection in that area. Since the overall power in that city is not sufficient, they have decided to build a power generator in that area and want to connect all the houses to the generator.

The area can be modeled as an **8 x 8** grid. Each cell contains one of the following characters

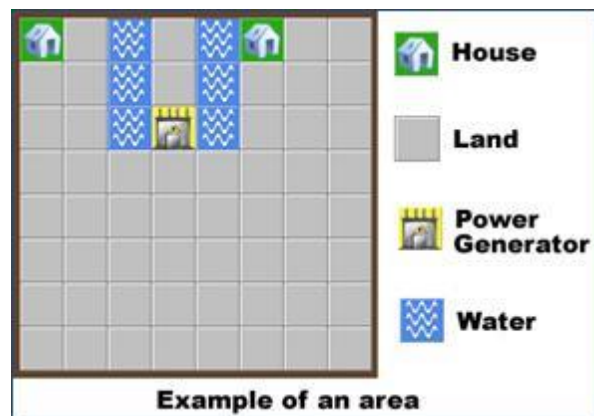
'.' means land

'H' means house

'G' means power generator

'W' means water

Two adjacent cells can be connected by cables and the cost is **1** thousand. Two cells are said to be adjacent if they share a side. But two adjacent cells can only be connected if none of the cells is empty. Empty means either land or water. In such case, pillars can be built in the cells and after that they can be connected. The cost of placing a pillar in a land and water cell is **pl** and **pw** thousand respectively. Remember that both the costs can be zero, because there can be sponsors who might use the pillars to advertise themselves.



Now given the modeled grid of the area, the PDB wants to find the minimum cost to connect all the houses to the power generator directly or indirectly. That's why they seek your help as you are one of the finest programmers in town.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case starts with two integers **pl** and **pw** ($0 \leq pl, pw \leq 10$). Then there will be **8** lines, each containing **8** characters from the set {'.', 'H', 'G', 'W'}. You may assume that in any modeled grid, there is exactly one power generator and the total number of houses is between **1** and **8** (inclusive).

Output

For each test case, print the case number and the minimum total cost in thousands.

Sample Input	Output for Sample Input
2 0 10 H.W.WH.. ..W.W.. ..WGW.. 0 0 H.W.WH.. ..W.W.. ..WGW..	Case 1: 12 Case 2: 7

SOLUTION

Can't find any!

1364 - Expected Cards

Time Limit: 3 second(s)

Memory Limit: 64 MB

Taha has got a standard deck of cards with him. In addition to the 52 regular ones, there are 2 joker cards. Every regular card has a rank and a suit. The ranks in ascending order are: **A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q** and **K**. The suit of a card can be **clubs, diamonds, hearts** or **spades**. That means there are 13 **clubs**, 13 **diamonds**, 13 **hearts** and 13 **spades** - which adds up to 52. The joker cards have no ranks or suits.

One day, Sara gave Taha a challenge. First she randomly shuffles the 54 cards and starts placing one card after another, face-up, on a table. What is the expected number of cards Sara has to place so that there are at least **C clubs, D diamonds, H hearts** and **S spades** on the table? Whenever a joker card is encountered, Taha has to assign it to some suit so that the expected number of cards to reach the goal is minimized. The decision of assigning the joker card to some suit has to be made instantly (i.e. before Sara puts the next card on the table). Note that the assignments of the two joker cards don't necessarily need to be the same.

Input

Input starts with an integer **T** (≤ 10), denoting the number of test cases.

Each case starts with a line containing four integers in the order **C, D, H** and **S**. Each of these integers will be in the range **[0, 15]**.

Output

For each case, print the case number first. Then output the expected number of cards Sara needs to place on the table to achieve the goal. If it's impossible to reach the goal, irrespective of what assignments Sara opts for, output **'-1'** (without the quotes) instead. Errors less than 10^{-6} will be ignored.

Sample Input	Output for Sample Input
4	Case 1: 0
0 0 0 0	Case 2: 54
15 13 13 13	Case 3: 16.3928186102
1 2 3 4	Case 4: -1
15 15 15 15	

Notes

1. For case 1, there is no need to place any card as all required values are 0
2. For case 2, we must place all the 54 cards to reach the goal
3. For case 3, note that output isn't always an integer
4. For case 4, 60 Cards? No way!!

SOLUTION

```
int cs , vis[16][16][16][16][6][6];
double dp[16][16][16][16][6][6] ;
int a[5];
double DP( int club , int dice , int hearts , int
spades , int j1 , int j2 )
{
    int c = ( 13 - club ) + ( j1 == 1 ) + ( j2 ==
1 );
    int d = ( 13 - dice ) + ( j1 == 2 ) + ( j2 ==
2 );
    int h = ( 13 - hearts ) + ( j1 == 3 ) + ( j2
== 3 );
    int s = ( 13 - spades ) + ( j1 == 4 ) + ( j2
== 4 );
    double tot = club + dice + hearts + spades +
( j1 == 0 ) + ( j2 == 0 );
    if( c >= a[0] && d >= a[1] && h >= a[2] && s
>= a[3] ) return 0.00 ;
    if( club + dice + hearts + spades + !j1 + !j2
== 0 ) return 10000000.000 ;
    int &v =
vis[club][dice][hearts][spades][j1][j2];
    if( v == cs ) return
dp[club][dice][hearts][spades][j1][j2];
    v = cs ;
    double ret = 0 ;
    if( club ) ret += ( club / tot ) * ( 1 + DP(
club - 1 , dice , hearts , spades , j1 , j2 ) );
    if( dice ) ret += ( dice / tot ) * ( 1 + DP(
club , dice - 1 , hearts , spades , j1 , j2 ) );
    if( hearts ) ret += ( hearts / tot ) * ( 1 + DP(
club , dice , hearts - 1 , spades , j1 , j2 ) );
    if( spades ) ret += ( spades / tot ) * ( 1 + DP(
club , dice , hearts , spades - 1 , j1 , j2 ) );
    double joker = ( j1 == 0 ) + ( j2 == 0 );
    double ans , tmp ;
    if( j1 == 0 )
    {
```

```

        ans = ( joker / tot ) * ( 1 + DP( club ,
dice , hearts , spades , 1 , j2 ) );
        tmp = ( joker / tot ) * ( 1 + DP( club ,
dice , hearts , spades , 2 , j2 ) );
        if( tmp < ans ) ans = tmp ;
        tmp = ( joker / tot ) * ( 1 + DP( club ,
dice , hearts , spades , 3 , j2 ) );
        if( tmp < ans ) ans = tmp ;
        tmp = ( joker / tot ) * ( 1 + DP( club ,
dice , hearts , spades , 4 , j2 ) );
        if( tmp < ans ) ans = tmp ;
        ret += ans ;
    }
    else if( j2 == 0 )
    {
        ans = ( joker / tot ) * ( 1 + DP( club ,
dice , hearts , spades , j1 , 1 ) );
        tmp = ( joker / tot ) * ( 1 + DP( club ,
dice , hearts , spades , j1 , 2 ) );
        if( tmp < ans ) ans = tmp ;
        tmp = ( joker / tot ) * ( 1 + DP( club ,
dice , hearts , spades , j1 , 3 ) );
        if( tmp < ans ) ans = tmp ;
        tmp = ( joker / tot ) * ( 1 + DP( club ,
dice , hearts , spades , j1 , 4 ) );
        if( tmp < ans ) ans = tmp ;
        ret += ans ;
    }
    return dp[club][dice][hearts][spades][j1][j2]
= ret ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i
will be a good human being
    cout << fixed << setprecision(10) ;
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        int ex = 0 ;

```

```

rep( i , 4 )
{
    a[i] = II ;
    if( a[i] > 13 ) ex += ( a[i] - 13 );
}
printf("Case %d: ",cs);
if( ex > 2 ) printf("-1\n");
else
{
    double res = DP( 13 , 13 , 13 , 13 ,
0 , 0 );
    cout << res << endl ;
}
}
return 0;
}

```

1365 - ICPC Guards

Time Limit: 2 second(s)

Memory Limit: 64 MB

This ICPC will take place in a huge hall room which can be divided into $N \times N$ square cells. That's why some volunteers will guard this room. But each row (or column) should be guarded by exactly two volunteers. And in a single cell at most one volunteer can be placed. Now volunteers can watch other volunteers either vertically or horizontally. Thus the volunteers form different groups. To be more specific, in a single group all the volunteers can look after each other directly or indirectly.

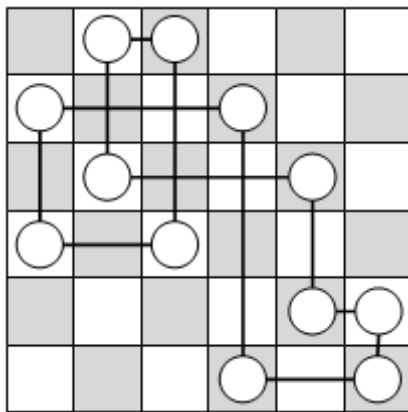


Fig 1

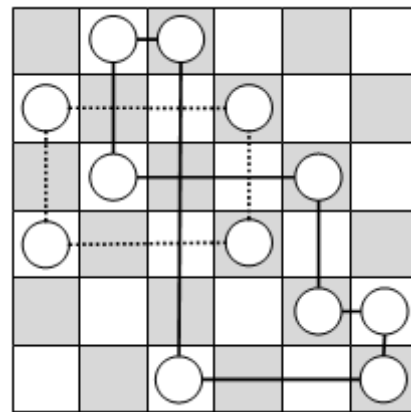


Fig 2

Suppose we have a hall room that can be divided into 6×6 square cells. Circles represent volunteers; lines represent the connectivity of the groups. In Fig 1, there is only one group (check the solid lines carefully). In Fig 2, there are two groups, one group is shown using solid lines, and another one is shown using dotted lines. Now the organizers wanted to know the number of ways they can place volunteers in the hall room such that they form exactly K groups. Two configurations will be different if in one configuration there is a volunteer on a cell but the cell is empty in another one. So, the organizers are seeking your help as you are one of the best programmers in town.

Input

Input starts with an integer **T** (≤ 50000), denoting the number of test cases.

Each case starts with a line containing two integers: **N** and **K** ($2 \leq N \leq 10^5$, $1 \leq K \leq \min(N, 50)$).

Output

For each case, print the case number and the number of ways the volunteers can be placed in the hall room as guards. The result can be large, so print the result modulo **1000 000 007**.

Sample Input	Output for Sample Input
4 2 1 3 1 4 1 4 2	Case 1: 1 Case 2: 6 Case 3: 72 Case 4: 18

SOLUTION

First of all i want to thanks Jan for such a beautiful problem.

This problem can be solved by dynamic programming. There are two approaches as far as i know and i am going to discuss the way i have solved it. You can get the judge solution analysis [here](#).

The approach i am going to discuss is another way of looking at the problem. Enough talk i think we should move on.

We are going to assume each column as the nodes of the graph. V is the set of the nodes of the graph, $V = \{v_1, v_2, v_3, \dots, v_n\}$ where v_i = represent i -th column. $E = \{(v_i, v_j) : v_i \text{ is not equals to } v_j\}$ is the set of edges of the graph and $L(v_i, v_j)$ = label of the edge (v_i, v_j) (the label of an edge will be row). The graph is directed . $R = \{r_1, r_2, \dots, r_n\}$ is the set of the rows where r_i = i -th row.

Oh god so boring lets start the fun part.

Suppose you have placed two guards at (r_1, c_1) and (r_1, c_2) . Then there will be an edge (c_1, c_2) and then you have placed another two at (r_3, c_2) and another one at (r_3, c_1) . So here we will get another edge (c_2, c_1) . So we have formed a cycle of length 2.

The cycle formed: $c_1 \rightarrow c_2 \rightarrow c_1$.

$L(c_1, c_2) = r_1$, as c_1 and c_2 are placed at row r_1 .

$L(c_2, c_1) = r_3$, as c_2 and c_1 are place at row r_3

$L(c_i, c_j)$ = the label of edge (c_i, c_j) is the row r_i if we place guards at (r_i, c_i) and (r_i, c_j) .

Now lets find answer for the simplest case.

lets say $n = 3$ and $k = 1$.

$$p! = p \cdot (p-1) \cdot \dots \cdot 1$$

so we have to form a cycle.

there are $(3-1)! = 2$ different cycles we can form of length three.

cycle 1: $c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_1$

cycle 2: $c_1 \rightarrow c_3 \rightarrow c_2 \rightarrow c_1$.

Now take cycle 1 and assign labels to each of the edges. There are $3!/2 = 3$ ways to assign them.

So for each cycle there are 3 ways to assign labels. So for both we have 6 ways and this is the answer for $n = 3$ and $k = 1$. That is there 6 possible ways we can place the guards maintaining conditions from the problem.

So the problem now modeled into a version of **stirling number** of first kind. Stirling number of first kind allows cycle of length 1 which we cannot allow.

we gonna define $S(n,k)$ as the number permutations of n elements with k disjoint cycles where each cycle length is greater than or equals to 2.

$S(n,k) = S(n-1,k) * (n-1) + S(n-2,k-1) * (n-1)$ [Find out how]

Result = $(S(n,k) * n!) / 2^k$.

as we have k cycles and we have n edges there $n!$ ways to level the edges and to remove over counting we have divided by 2^k .

Thanks

This is my very first post and pardon me for any mistake :)

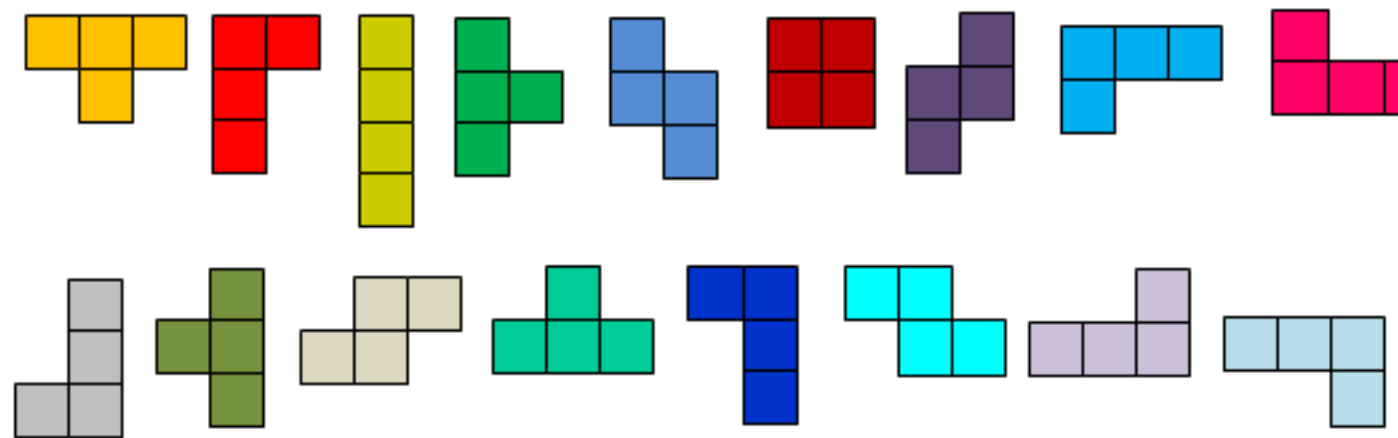
1376 - Tetromino

Time Limit: 5 second(s)

Memory Limit: 32 MB

Dexter wants to completely cover a $4 \times N$ board using N tetrominoes. Every cell in the grid has to be covered by exactly one piece and no piece is allowed to go outside the board. And no piece can be rotated or flipped.

A tetromino is a connected set of 4 tiles. All 19 possible tetrominoes that can be used (any number of times) are shown below:



Assume that all 19 pieces have different colors. Now your task is to find the total number of ways Dexter can cover the board. Two board configurations are different if a cell can be found where the colors are different.

Input

Input starts with an integer T (≤ 20), denoting the number of test cases.

Each case starts with a line containing an integer N ($1 \leq N \leq 10^9$).

Output

For each case, print the case number and the number of ways to fill the board modulo 1000 000 007.

Sample Input	Output for Sample Input
4	Case 1: 1
1	Case 2: 4
2	Case 3: 23
3	Case 4: 15747348
12	

SOLUTION

Can't find any!

1381 - Scientific Experiment

Time Limit: 2 second(s)

Memory Limit: 32 MB

John wants to be a scientist. A first step of becoming a scientist is to perform experiment. John has decided to experiment with eggs. He wants to compare the hardness of eggs from different species. He has decided to use a nearby large multi-storied building for this purpose. For each species he will try to find the highest floor from which he can drop the egg and it will not break. The building has $(n+1)$ floors numbered from 0 to n . John has a book from which he knows that



1. If an egg is dropped from the topmost floor, it will surely break.
2. If an egg is dropped from floor 0 , it will not break.
3. The eggs of same species are of same strength. That means if any egg breaks when dropped from the k^{th} floor; all the eggs of that species will break if dropped from k^{th} floor.
4. If an egg is unbroken after dropping it from any floor, it remains unharmed, that means the strength of the egg remains same.

Unfortunately John has a few problems:

1. He can only carry one egg at a time.
2. He can buy eggs from a shop inside the building and an egg costs x cents.
3. To enter the building he has to pay y cents if he has no egg with him and z cents if he carries an egg with him.
4. After dropping an egg, John must go outside the building to check whether it's broken or not.
5. He does not want to waste any egg so he will not leave any unbroken egg on the ground. But if an egg is broken, he leaves it there.
6. If he has an intact egg at the end, he can sell it for $x/2$ cents. He does not need to enter the building to sell the egg.

These problems are not going to tame John's curious mind. So he has decided to use an optimal strategy and minimize his cost in worst case. As John is not a programmer, he asked your help.

Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each case starts with a line containing four integers n x y z as described in the statement. You may assume that $1 < n \leq 1000$ and $1 \leq x, y, z \leq 10^5$ and x is even.

Output

For each test case, print the case number and the minimized worst case cost.

Sample Input	Output for Sample Input
7	Case 1: 2000
4 2 998 1000	Case 2: 4008
16 2 1000 1000	Case 3: 1015
16 1000 1 1	Case 4: 1003
4 1000 1 1	Case 5: 10
7 2 2 2	Case 6: 24
9 2 1 100	Case 7: 111
11 2 100 1	

Note

For case 1, John knows that the egg will break if dropped from 4th floor, but will not break if dropped from 0th floor. An optimal solution may be

1. John enters the building without any egg (€998).
2. John buys an egg (€2).
3. John drops an egg from 2nd floor. John goes out and checks the egg.
 - a. If it breaks,
 - i. John again enters the building without any egg (€998) and buys an egg there €2.
 - ii. He drops the egg from 1st floor.
 1. If it does not break then answer to his problem is 1 and he can sell the egg for €1. So his final cost is €1999.
 2. If it breaks then the answer to his problem is 0th floor and his final cost is €2000.
 - b. If it does not break
 - i. John enters the building with the egg (€1000).
 - ii. He drops it from 3rd floor.
 1. If it does not break then answer to his problem is 3 and he can sell the egg for €1. So his final cost is €1999.
 2. If it breaks then the answer to his problem is 2 and final cost is €2000.

So, using this strategy, his worst case cost is €2000.

SOLUTION

```
//qscqesze
#include <cstdio>
#include <cmath>
#include <cstring>
#include <ctime>
#include <iostream>
#include <algorithm>
#include <set>
#include <vector>
#include <sstream>
#include <queue>
#include <typeinfo>
#include <fstream>
#include <map>
#include <stack>
typedef long long ll;
using namespace std;
//freopen("D.in","r",stdin);
//freopen("D.out","w",stdout);
#define sspeed ios_base::sync_with_stdio(0);cin.tie(0)
#define maxn 200001
#define mod 10007
#define eps 1e-9
int Num;
char CH[20];
//const int inf=0x7fffffff; //нечёт||C
```

```

const int inf=0x3f3f3f3f;

/*

inline void P(int x)
{
    Num=0;if(!x){putchar('0');puts("");return;}
    while(x>0)CH[++Num]=x%10,x/=10;
    while(Num)putchar(CH[Num--]+48);
    puts("");
}
*/

inline ll read()
{
    ll x=0,f=1;char ch=getchar();
    while(ch<'0' || ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
    return x*f;
}

inline void P(int x)
{
    Num=0;if(!x){putchar('0');puts("");return;}
    while(x>0)CH[++Num]=x%10,x/=10;
    while(Num)putchar(CH[Num--]+48);
    puts("");
}

// *****
***

ll dp[maxn];

```

```

ll d1,d2;

int n,x,y,z;

int main()
{
    int t=read();
    for(int cas=1;cas<=t;cas++)
    {
        n=read(),x=read(),y=read(),z=read();
        dp[0]=dp[1]=0;
        for(int i=2;i<=n;i++){
            dp[i]=inf;
            for(int j=1;j<i;j++)
            {
                d1=dp[j]+x+y;
                d2=dp[i-j]+z;
                if(i-j==1)
                    d2 = y + x / 2 ;
                dp[i]=min(dp[i],max(d1,d2));
            }
        }
        printf("Case %d: %lld\n",cas,dp[n]);
    }
}

```

1382 - The Queue

Time Limit: 2 second(s)

Memory Limit: 32 MB

On some special occasions Nadia's company provide very special lunch for all employees of the company. Before the food is served all of the employees must stand in a queue in front of the food counter. The company applied a rule for standing in the queue. The rule is nobody can stand anywhere in front of his supervisor in the queue. For example, if Abul is the supervisor of Babul and Abul stands in k^{th} position from the front of the queue, then Babul cannot stand at any position in between 1 and $k - 1$ from front of the queue.

The company has N employees and each of them has exactly one supervisor except one (CEO) who doesn't have any supervisor.

You have to calculate in how many ways the queue can be created. For this problem, you can safely assume that in at least one way the queue can be created.

Input

Input starts with an integer T (≤ 700), denoting the number of test cases.

Each case starts with a line containing an integer N ($1 \leq N \leq 1000$). Each of the following $N - 1$ lines will contain two integers a and b ($1 \leq a, b \leq N, a \neq b$), which denotes that a is the supervisor of b . For the sake of simplicity we are representing each employee by an integer number. Assume that the given input follows the restrictions stated above.

Output

For each case, print the case number and the number of ways to create the queue. The result can be large, print the result modulo 1000 000 007.

Sample Input	Output for Sample Input
1 5 2 1 2 3 3 4 3 5	Case 1: 8

SOLUTION


```
const int NX = 1005 ;
const Long mod = 1000000007;
Long nCr[NX][NX] , dp[NX] , deg[NX] , adj[NX][NX] , hasChild[NX] ,
notroot[NX] ;
int vis[NX] , cs ;
void pre()
{
    int i , j ;
    rep( i , NX ) nCr[i][0] = 1 ;
    for( i = 1 ; i < NX ; i++ )
    {
        for ( j = 1 ; j <= i ; j++ ) nCr[i][j] = ( nCr[i-1][j-1] +
nCr[i-1][j] )%mod;
    }
}
Long childCount(int x)
{
    if( deg[x] == 0 ) return hasChild[x] = 1 ;
    hasChild[x] = 1 ;
    rep( i , deg[x] )
    {
        int v = adj[x][i];
        hasChild[x] += childCount(v);
    }
    return hasChild[x];
}
Long DP( int x )
{
    if( vis[x] == cs ) return dp[x] ;
    vis[x] = cs ;
    if( deg[x] == 0 ) return dp[x] = 1 ;
    dp[x] = 1 ;
    Long totalchild = hasChild[x] - 1; // except himself
    rep( i , deg[x] )
    {
        int v = adj[x][i] ;
        dp[x] = ( ( dp[x] * nCr[totalchild][hasChild[v]])%mod *
DP(v))%mod;
```

```

        totalchild -= hasChild[v];
    }
    return dp[x] ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human
    being
    // cout << fixed << setprecision(10) ;
    pre();
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        int n = II ;
        rep ( i , n )
        {
            notroot[i] = 0 ;
            deg[i] = 0 ;
        }
        rep( i , n -1 )
        {
            int x = II , y = II ;
            x-- , y--;
            adj[x][deg[x]++] = y ;
            notroot[y] = 1 ;
        }
        rep( i , n )
        {
            if( notroot[i] ) continue ;
            childCount(i);
            printf("Case %d: %lld\n",cs,DP(i)%mod);
            break;
        }
    }
    return 0;
}

```

1394 - Disable the Wand

SUBMIT 	PDF (English)	Statistics	Forum
Time Limit: 4 second(s)		Memory Limit: 32 MB	

The battle of Hogwarts is going to start very soon. Hermione has reviewed the whole strategy of the battle and she finds that they need to disable some wands of some death eaters. It is very difficult to perform magic without wands, so they will have some significant advantages over those death eaters. She quickly discovers that to disable a wand they have to know the sum of the magical numbers corresponding to that wand.

The properties of a magical number of a wand are given below:

1. The number must be greater than or equal to a certain number **start**.
2. The number must be less than or equal to a certain number **end**.
3. The binary representation of a magical number contains at most **Maxone** number of ones.
4. The binary representation of a magical number can differ from **Ideal Number** at not more than **k** positions. For example, 0110_2 (6_{10}) differs from 1010_2 (10_{10}) at two positions.
5. A magical number must be an integer which is divisible by 3, but not divisible by 7.

As both start and end can be quite large, she needs your help to find the sum of the magical numbers within this range.

Input

Input starts with an integer **T** (≤ 130), denoting the number of test cases.

For each case, a single line follows which contains five integers: **start**, **end**, **Maxone**, **Ideal Number**, **k**, respectively. All of them are non-negative integers and less than or equal to 10^9 . And **start** will not be greater than **end**.

Output

For each case, print the case number and sum of the magical numbers.

Sample Input	Output for Sample Input
2 1 6 2 3 1 1 6 2 3 2	Case 1: 3 Case 2: 9

Note

For computing the number of positions of difference from the ideal number, both ideal number and magical number can be considered as 32 bit binary numbers with necessary number of leading zeros.

SOLUTION

```
const int MX = 35 ;
Long st , en , maxone , maxdiff , ideal , A ;
pll dp[MX][MX][MX][3][7][2];
bool vis[MX][MX][MX][3][7][2];
pll DP(int idx , int one , int diff , int mod3 , int mod7 , int isSmall
)
{
    // printf("")
    if( diff > maxdiff || one > maxone ) return mp(0,0);
    if( idx < 0 )
    {
        // printf("mod3 :: %d mod7::%d\n",mod3,mod7);
        if( !mod3 && mod7 ) return make_pair(011,111);
        else return mp(011,011);
    }
    pll &ret = dp[idx][one][diff][mod3][mod7][isSmall];
    if( vis[idx][one][diff][mod3][mod7][isSmall]) return ret ;
    vis[idx][one][diff][mod3][mod7][isSmall] = 1 ;
    // ret = mp(0,0);
    pll tmp , rt ;
    int cur = (ideal & ( 111 << idx )) ? 1 : 0 ;
    if( isSmall || (A & ( 111 << idx )) )
    {
        // printf("here\n");
        // 1 bosai
        tmp = DP(idx-1,one+1 , diff + ( cur ^ 1 ) , ( mod3 + ( 111 <<
idx ))%311 , (mod7 + ( 111 << idx))%711 , isSmall );
        rt.ff += (( 111 << idx)*tmp.ss) + tmp.ff ;
        rt.ss += tmp.ss ;
        // o bosai
        // cout << ret.ff << " " << tmp.ss << " " << endl ;
        tmp = DP(idx-1,one,diff+cur,mod3,mod7,1);
        rt.ff += tmp.ff ;
        rt.ss += tmp.ss ;
    }
    else
    {
        tmp = DP(idx-1,one,diff+cur,mod3,mod7,isSmall);
```

```

        rt.ff += tmp.ff ;
        rt.ss += tmp.ss ;
    }
    // printf("idx :: %d == ",idx);
    // cout << rt.ff << endl ;
    return ret = rt ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human
    being
    int cs , t = II ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        st = LL , en = LL , maxone = LL , ideal = LL , maxdiff = LL ;
        //ms(dp,-1);
        ms(vis,0);
        A = en ;
        pll two = DP(32,0,0,0,0,0) ;
        if( st == 0 ) printf("Case %d: %lld\n",cs,two.ff);
        else
        {
            A = st - 1;
            ms(vis,0);
            pll one = DP(32,0,0,0,0,0) ;
            // cout << one.ff << " " << two.ff << endl ;
            printf("Case %d: %lld\n",cs,two.ff-one.ff);
        }
    }
    return 0;
}

```

1399 - Politeness

Time Limit: 4 second(s)

Memory Limit: 32 MB

A number is called polite if it can be written as a sum of some (at least two) consecutive positive integers. For example, 6 (1+2+3) is a polite number while 4 is not. Politeness of a number is the number of ways it can be expressed as sum of consecutive positive integers. For example 6(1+2+3) is a politeness of 1 while 18(5+6+7 = 3+4+5+6) has a politeness of 2. Obviously, non polite number has a politeness of 0.

You are given a politeness **P**, you have to find out the smallest number that has the politeness **P**.

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case starts with a line containing an integer **P** ($0 \leq P \leq 10^{12}$).

Output

For each case, print the case number and the integer that has the politeness **P**. If there are several integers having politeness **P**, choose the smallest one. As the integer can be big; print the integer modulo 1000 000 007.

Sample Input	Output for Sample Input
7	Case 1: 1
0	Case 2: 3
1	Case 3: 45
5	Case 4: 343299432
852281891999	Case 5: 129653419
975018442254	Case 6: 221997673
986350945007	Case 7: 3917908
511	

Note

For case 7, 1003917915 is the smallest number with politeness 511, so the result is (1003917915 modulo 1000000007), which is 3917908.

SOLUTION

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <vector>
#include <algorithm>
#define i64 long long
#define u32 unsigned int
using namespace std;

int C,num=0;

const int MAX=1000005;
u32 day [MAX / 64 + 5];
int prime[MAX/10],cnt;

void init()
{
    prime[cnt++] = 2;
    int i,j;
    for(i=3;i<=MAX;i+=2) if(!(tag[i]>>6] & (1<<(i/2%32))))
    {
        prime[cnt++] = i;
        for(j=i*3;j<=MAX;j+=i*2) tag[j]>>6] |= (1<<(j/2%32));
    }
}
```



```
const i64 MOD=1000000007;
```

```
i64 n;
```

```
vector<i64> V;
```

```
i64 POW(i64 a,i64 b)
```

```
{
```

```
    i64 years = 1;
```

```
    while(b)
```

```
    {
```

```
        if(b&1) ans=ans*a%MOD;
```

```
        a=a*a%MOD;
```

```
        b >> = 1;
```

```
    }
```

```
    return years;
```

```
}
```

```
double Min;
```

```
i64 years old;
```

```
struct node
```

```
{
```

```
    double f;
```

```
    i64 p,qflag;
```

```
};
```

```
node a[150];
```

```

int cmp(node a,node b)
{
    return a.f<b.f;
}

double f[2][50];
i64 q[2][4][50],p[2][50];
int pre,cur;

void DFS(int dep,int stateNum)
{
    int i,j,k,t=0,r,c;
    if(dep==V.size())
    {
        for(i=1;i<=stateNum;i++) if(f[pre][i]<Min)
        {
            My = f [pre] [i];
            ans = p [pre] [i];
        }
        return;
    }
    i64 x=V[dep];
    for(i=1;i<=stateNum;i++) for(j=1;j<=V.size();j++)
    {
        t++;
        a[t].qflag=i*100+j;
        a[t].f=f[pre][i]+q[pre][i][j]*(x-1)*log(prime[j]*1.0);
        a[t].p=p[pre][i]*POW(prime[j],q[pre][i][j]*(x-1))%MOD;
    }
}

```

```

sort(a+1,a+t+1,cmp);
k=0;
for(i=1;i<=t&& k<3;i++) if(i==1||a[i].p!=a[i-1].p)
{
    k++;
    f[cur][k]=a[i].f;
    p[cur][k]=a[i].p;
    r=a[i].qflag/100;
    c=a[i].qflag%100;
    for(j=1;j<=V.size();j++) q[cur][k][j]=q[pre][r][j];
    q[cur][k][c]*=x;
}
pre^=1;
cur^=1;
DFS(dep+1,k);
}

```

```

i64 get()
{
    if(n==1) return 1;
    i64 i,j,k;
    V.clear();
    for(i=0;i<cnt&&(i64)prime[i]*prime[i]<=n;i++)
    {
        while(n%prime[i]==0)
        {
            V.push_back(prime[i]);
            n/=prime[i];
        }
    }
}

```

```

    }
    if(n>1) V.push_back(n);
    for(i=0,j=V.size()-1;i<j;i++,j--)
    {
        k = V [i];
        W [i] = V [j];
        W [j] = k;
    }
    Min = 1e40;
    for(i=1;i<=V.size();i++) q[0][1][i]=1;
    f[0][1]=0;
    p[0][1]=1;
    pre=0;cur=1;
    DFS(0,1);
    return years;
}

int main()
{
    init();
    for(scanf("%d",&C);C--;)
    {
        scanf("%lld",&n);
        n++;
        printf("Case %d: %lld\n",++num,get());
    }
    return 0;
}

```

1406 - Assassin's Creed

Time Limit: 4 second(s)

Memory Limit: 32 MB

Altair is in great danger as he broke the three tenets of the assassin creed. The three tenets are: 1) never kill an innocent people, 2) always be discrete and 3) never compromise the brotherhood. As a result Altair is given another chance to prove that he is still a true assassin. Altair has to kill n targets located in n different cities. Now as time is short, Altair can send a message along with the map to the assassin's bureau to send some assassins who will start visiting cities and killing the targets. An assassin can start from any city, but he cannot visit a city which is already visited by any other assassin except him (because they do not like each other's work). He can visit a city multiple times though. Now Altair wants to find the minimum number of assassins needed to kill all the targets. That's why he is seeking your help.



Input

Input starts with an integer T (≤ 50), denoting the number of test cases.

Each case starts with a blank line. Next line contains two integers n ($1 \leq n \leq 15$) and m ($0 \leq m \leq 50$), where n denotes the number of cities and m denotes the number of one way roads. Each of the next m lines contains two integers u v ($1 \leq u, v \leq n, u \neq v$) meaning that there is a road from u to v . Assume that there can be at most one road from a city u to v .

Output

For each case, print the case number and the minimum number of assassins needed to kill all the targets.

Sample Input	Output for Sample Input
2	Case 1: 1
3 2	Case 2: 2
1 2	
2 3	
6 6	
1 2	
2 3	
2 4	
5 4	
4 6	
4 2	

SOLUTION

```
const int INF = 1 << 29 ;
int sv[20] , dp[ ( 1 << 16 ) + 10 ] , vis[ ( 1 << 16 ) + 10 ] , cs ;
int deg[17] , adj[20][20] , mem[17][ 1 << 17 ] , isPossible[ 1 << 17 ] ,
n , m ;
int rdeg[17] , rev[20][20];
void dfs( int x , int mask )
{
    mem[x][mask] = 1 ;
    isPossible[mask] = 1 ;
    rep( i , deg[x])
    {
        int u = adj[x][i];
        if( !mem[u][mask|(1 << u)]) dfs( u , mask | ( 1 << u ) );
    }
}
int DP( int mask )
{
    if( mask == 0 ) return 0 ;
    int &v = vis[mask];
    int &ret = dp[mask];
    if( v == cs ) return ret ;
    v = cs ;
    ret = INF ;
    int i ;
    for ( i = mask ; i ; i = ( mask ) & ( i - 1 ) )
    {
        if( isPossible[i] ) ret = min( ret , 1 + DP( mask ^ i ) );
    }
    return ret ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human
    being
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++)
    {
```

```

n = II , m = II ;
ms( deg , 0 );
ms( mem , 0 );
ms( isPossible , 0 );
rep( i , m )
{
    int x = II , y = II ;
    x-- , y--;
    adj[x][deg[x]++] = y ;
}
rep( i , n ) dfs( i , 1 << i );
printf("Case %d: %d\n",cs,DP(( 1 << n ) - 1 ));
}
return 0;
}

```


1415 - Save the Trees

Time Limit: 5 second(s)

Memory Limit: 64 MB

N trees were planted on a side of a long straight road. In the other side there are farms, industries etc. The market price of these trees is huge now. Some greedy people want to cut these trees down and want to be millionaires. They got the permission from the Govt. decoying that they would develop the area. You published this fact in the web and millions raised their voices against this conspiracy.

You gathered the information of the trees and found the **type** and **height** of all trees. For simplicity, you represented them as integers. You want to find the overall price of the trees. To find the price, the following method is used.

- 1) The trees are first partitioned into groups with the condition that, types of two trees will not be similar in a group. A group can only be formed using contiguous trees.
- 2) The price of a group is equal to the height of the tallest tree.
- 3) The overall price is the summation of prices of all groups.

Now you want to find the minimum possible price of the trees in this scheme and show the Govt. that even though you calculated the minimum possible price, it's actually huge.

Input

Input starts with an integer **T** (≤ 5), denoting the number of test cases.

Each case starts with an integer **N** ($1 \leq N \leq 2*10^5$). Each of the next **N** lines contains two integers **type_i** **height_i** ($1 \leq \text{type}_i, \text{height}_i \leq 2*10^5$) of the **ith** tree from left to right.

Output

For each case, print the case number and the minimum possible price of the trees according to the scheme described above.

Sample Input	Output for Sample Input
2 5 3 11 2 13 1 12 2 9 3 13 4 3 5 2 21	Case 1: 26 Case 2: 31

3 12	
2 5	

Note

Dataset is huge, use faster I/O methods.

SOLUTION

```
const int NX = 2e5 + 10 ;
const Long INF = 1e17 + 10 ;
Long tree[(4*NX)] , Lazy[(4*NX)] ;
int prv[NX] , last[NX] , typed[NX] ; Long H[NX] ;
struct abc
{
    int lft , rgt ; Long height ;
    abc() {}
    abc( int a , int b , Long c)
    {
        lft = a ;
        rgt = b ;
        height = c ;
    }
};
stack < abc > st ;
void LazyPropagate( int node , int lft , int rgt)
{
    tree[node] += Lazy[node];
    if( lft < rgt )
    {
        Lazy[2*node] += Lazy[node];
        Lazy[(2*node) + 1 ] += Lazy[node];
    }
    Lazy[node] = 0 ;
}
Long update( int node , int lft , int rgt , int a , int b , Long val )
{
    if( Lazy[node] ) LazyPropagate(node,lft,rgt);
    if( b < lft || a > rgt ) return tree[node];
    if( a <= lft && b >= rgt )
    {
        tree[node] += val ;
        if( lft != rgt )
        {
            Lazy[2*node] += val ;
            Lazy[(2*node)+1] += val ;
        }
    }
}
```

```

        return tree[node];
    }
    int mid = ( lft +rgt )/2 ;
    tree[node] = min( update(2*node,lft,mid,a,b,val) , update((2*node) +
1 , mid + 1 , rgt , a , b , val));
    return tree[node];
}
Long query( int node , int lft , int rgt , int a , int b)
{
    if( Lazy[node] ) LazyPropagate(node,lft,rgt);
    if( b < lft || a > rgt ) return INF;
    if( a <= lft && b >= rgt )
    {
        return tree[node];
    }
    int mid = ( lft +rgt )/2 ;
    Long ans = min( query(2*node,lft,mid,a,b) , query((2*node) + 1 , mid
+ 1 , rgt , a , b ));
    return ans;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good human
being
    int t = II , cs ;
    for ( cs = 1 ; cs <= t ; cs++ )
    {
        ms( tree , 0 );
        ms( Lazy , 0 );
        ms( last , 0 );
        int n = II ;
        For( i , n )
        {
            typed[i] = II , H[i] = LL ;
            prv[i] = last[typed[i]];
            last[typed[i]] = i ;
        }
        while( !st.empty() ) st.pop();
        st.push(abc(-1,-1,INF));
        int L = 0 , R = -1 ;

```

```

Long ans ;
For( i , n )
{
    L = max( L , prv[i] );
    R++;
    while( st.top().height < H[i] )
    {
        update(1 , 0 , n , st.top().lft , st.top().rgt , -
st.top().height );
        st.pop();
    }
    st.push(abc(st.top().rgt + 1 , R , H[i] ) );
    update( 1 , 0 , n , st.top().lft , st.top().rgt ,
st.top().height );
    ans = query( 1 , 0 , n , L , R );
    update( 1 , 0 , n , R + 1 , R + 1 , ans );
}
printf("Case %d: %lld\n",cs,ans);
}
return 0;
}

```

1416 - Superb Sequence

Time Limit: 3 second(s)

Memory Limit: 64 MB

There were three friends (Alice, Bob and Carol) who regularly went to expeditions and discovered new mountain peaks. They often proposed different names and it was a problem to decide which name they would choose for the newly discovered peaks. Alice and Bob both said that the name of the peak must be a super sequence of their proposed names **A** and **B**, i.e. **A** and **B** should be **subsequences** of the name of the peak. Carol said that the name of the peak must be a **subsequence** of her proposed name **C**. As they don't like long names, they want to know the number of distinct shortest names which satisfy their needs.

So, given three strings **A**, **B** and **C**, you have to find the number of distinct shortest **common super sequences** of **A** and **B** who are also a **subsequence** of **C**. Moreover, you need to find the lexicographically earliest such sequence. Two sequences are distinct if they differ in at least one position. A **subsequence** is a sequence obtained by deleting zero or more characters from a string. A **super-sequence** is a sequence obtained by inserting zero or more characters in one or more positions of the string.

For example, say, **A** = "cdfa", **B** = "dga" and **C** = "bcdfgaga". Then there are two shortest common super sequences of **A** and **B**: "cdfga" and "cdgfa", but "cdgfa" is not a subsequence of **C**. So the only possible name for the peak is "cdfga".

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case contains three lines. First line contains a string denoting **A**, second line contains **B** and third line contains **C**. Assume that the strings are non-empty and length of **A** and **B** will not be more than 100 and length of **C** will not be more than 300.

Output

For each case, print the case number and the number of distinct possible shortest names for the peak modulo 1000 000 007. And second line should contain the lexicographically earliest name. If no solution is found then print "NOT FOUND" in second line.

Sample Input	Output for Sample Input
2 cdfa dga bcdfgaga	Case 1: 1 cdfga Case 2: 0 NOT FOUND

abc	
defm	
abcdefghijklm	

SOLUTION

```
string a,b,c;
pair<int,int> dp[105][105][305];
int don[105][105][305]={0},cc=1;
int x[105][30];
int y[105][30];
int z[305][30];
pair<int,int> re(int l,int r,int p)
{
    // pf("%d %d %d\n",l,r,p);
    if( (sz(a)>1 || sz(b)>r) && p==sz(c) ) return mp(1,0);
    if(sz(a)==1 && sz(b)==r) return mp(0,1);
    pair<int,int> &res=dp[l][r][p];
    if(don[l][r][p]==cc) return res;
    don[l][r][p]=cc;
    res=mp(1,0);
    pair<int,int> g;
    for(int i=0;i<26;i++)
    {
        if(x[l][i]!=-1 && y[r][i]!=-1 && z[p][i]!=-1 )
        {
            g=re(x[l][i]+1,y[r][i]+1,z[p][i]+1);
            g.F++;
            if(g.F==res.F)
                res.S=(res.S+g.S);
            else if(res.F>g.F)res=g;
        }
        // what_is(re(x[l][i]+1,y[r][i],z[p][i]+1))
        else if(x[l][i]!=-1 && z[p][i]!=-1 )
        {
            g=re(x[l][i]+1,r,z[p][i]+1);
            g.F++;
            if(g.F==res.F)
                res.S=(res.S+g.S);
            else if(res.F>g.F)res=g;
        }
        else if(y[r][i]!=-1 && z[p][i]!=-1 )
        {
            g=re(l,y[r][i]+1,z[p][i]+1);
```



```

        g.F++;
        if(g.F==res.F)
            res.S=(res.S+g.S);
        else if(res.F>g.F)res=g;
    }
}
res.S=mod_v(res.S);
return res;
}
string s;
void fuck(int l,int r,int p,int val)
{
    if(val==0) return ;
    bool ok=true;
    for(int i=0;i<26;i++)
    {
        if(x[l][i]!=-1 && y[r][i]!=-1 && z[p][i]!=-1 &&
val==1+re(x[l][i]+1,y[r][i]+1,z[p][i]+1).F )
        {
            if(ok)
            {
                s.pb(i+'a');
                ok=false;
            }
            fuck(x[l][i]+1,y[r][i]+1,z[p][i]+1,val-1);
            return;
        }
//        what_is(re(x[l][i]+1,y[r][i],z[p][i]+1))
        if(x[l][i]!=-1 && z[p][i]!=-1 &&
val==1+re(x[l][i]+1,r,z[p][i]+1).F )
        {
            if(ok)
            {
                s.pb(i+'a');
                ok=false;
            }
            fuck(x[l][i]+1,r,z[p][i]+1,val-1);
            return;
        }
        if(y[r][i]!=-1 && z[p][i]!=-1 &&
val==1+re(l,y[r][i]+1,z[p][i]+1).F )

```

```

        {
            if(ok)
            {
                s.pb(i+'a');
                ok=false;
            }
            fuck(1,y[r][i]+1,z[p][i]+1,val-1);
            return;
        }
    }
}

int main()
{
    //      std::ios_base::sync_with_stdio(false);
    #ifdef kimbbakar
        freopen ( "in.txt", "r", stdin );
        freopen ( "out.txt", "w", stdout );
    #endif
    int t,tcase=1;
    in(t);
    while(t--)
    {
        cin>>a>>b>>c;
        reset(x,-1);
        reset(y,-1);
        reset(z,-1);
        for(int i=sz(a)-1;i>=0;i--)
        {
            for(int j=0;j<26;j++)
                x[i][j]=z[i+1][j];
            x[i][a[i]-'a']=i;
        }
        for(int i=sz(b)-1;i>=0;i--)
        {
            for(int j=0;j<26;j++)
                y[i][j]=z[i+1][j];
            y[i][b[i]-'a']=i;
        }
        for(int i=sz(c)-1;i>=0;i--)
        {
            for(int j=0;j<26;j++)

```

```

        z[i][j]=z[i+1][j];
        z[i][c[i]-'a']=i;
    }
    s.clear();
    pair<int,int> g=re(0,0,0);
    pf("Case %d: ",tcase++);
    if(g.F==inf)
    {
        pf("\nNOT FOUND\n");
    }
    else
    {
        pf("%d\n",g.S);
        fuck(0,0,0,g.F);
        cout<<s<<"\n";
    }
    cc++;
}
return 0;
}

```

1420 - Subsequences forming Strings

Time Limit: 4 second(s)

Memory Limit: 32 MB

Given three strings **A**, **B** and **C** you have to count the number of ways you can construct **C** by combining two **subsequences** from **A** and **B**.

After deleting 0 or more characters from a string we can get its **subsequence**. For example "a", "b", "c", "ab", "ac", "bc", "abc", "" (empty string) are the subsequences of "abc".

Now, suppose there are two subsequences "abc" and "de". By combining them you can get the following strings "abcde", "abdce", "abdec", "adbce", "adbec", "adebc", "dabce", "dabec", "daebc" and "deabc".

Input

Input starts with an integer **T** (≤ 100), denoting the number of test cases.

Each case starts with a line containing three strings **A**, **B** and **C**. The strings will contain only lowercase letters and the lengths of the strings are between 1 and 100 (inclusive).

Output

For each case, print the case number and the number of ways you can construct **C** from the first two strings: **A** and **B** by the above way. The result can be large, so print the result modulo 1000000007.

Sample Input	Output for Sample Input
2 abc abc abc abbcd bccde abcde	Case 1: 8 Case 2: 18

SOLUTION

```
const int mod = 1000000007;
const int NX = 102 ;
int dp[NX][NX][NX][2] , lena , lenb , lenc , cs ,
vis[NX][NX][NX][2] ;
char A[NX] , B[NX] , C[NX];
int DP( int a , int b , int c , int which)
{
    if( c == lenc ) return 1 ;
    if( a == lena && b == lenb ) return 0 ;
    int &ret = dp[a][b][c][which];
    int &v = vis[a][b][c][which];
    if( v == cs ) return ret ;
    v = cs ;
    ret = 0 ;
    if( a < lena && !which)
    {
        ret += DP( a + 1 , b , c , 0 );
        ret %= mod ;
        if( A[a] == C[c] )
        {
            ret += DP( a + 1 , b , c + 1 , 0 );
            ret %= mod ;
            if( c + 1 < lenc )
            {
                ret += DP( a + 1 , b , c + 1 , 1 );
                ret %= mod ;
            }
        }
    }
    if( b < lenb && which )
    {
        ret += DP( a , b + 1 , c , which );
        ret %= mod ;
        if( B[b] == C[c] )
        {
            ret += DP( a , b + 1 , c + 1 , which );
            ret %= mod ;
            if( c + 1 < lenc )
```

```

        {
            ret += DP( a , b + 1 , c + 1 , !which );
            ret %= mod ;
        }
    }
}
return ret ;
}
int main()
{
    // I will always use scanf and printf
    // May be i won't be a good programmer but i will be a good
human being
    int t = II ;
    for ( cs = 1 ; cs <= t ; cs++)
    {
        scanf("%s%s%s",A,B,C);
        lena = strlen(A);
        lenb = strlen(B);
        lenc = strlen(C);
        int ans = ( DP( 0 , 0 , 0 , 0 ) + DP( 0 , 0 , 0 , 1 )
)%mod ;
        printf("Case %d: %d\n",cs,ans);
    }
    return 0;
}

```

1421 - Wavio Sequence

Time Limit: 4 second(s)

Memory Limit: 32 MB

Wavio is a sequence of integers. It has some interesting properties:

1. Wavio is of odd length i.e. $L = 2*n + 1$.
2. The first $(n+1)$ integers of Wavio sequence make a strictly increasing sequence.
3. The last $(n+1)$ integers of Wavio sequence make a strictly decreasing sequence.
4. No two adjacent integers are same in a Wavio sequence.

For example 1, 2, 3, 4, 5, 4, 3, 2, 1 is an Wavio sequence of length 9. But 1, 2, 3, 4, 5, 4, 3, 2, 2 is not a valid wavio sequence. In this problem, you will be given a sequence of integers. You have to find the length of the longest Wavio sequence which is a **subsequence** of the given sequence. Consider the given sequence as:

1 2 3 2 1 2 3 4 3 2 1 5 4 1 2 3 2 2 1

Here the longest Wavio sequence is: 1 2 3 4 5 4 3 2 1. So, the output will be 9.

Input

Input starts with an integer T (≤ 12), denoting the number of test cases.

Each case starts with a line containing an integer N ($1 \leq N \leq 10^5$) denoting the number of elements in the sequence. The next line contains N space separated integers between -10^8 to 10^8 , that form the sequence.

Output

For each case, print the case number and the length of the maximum possible Wavio sequence.

Sample Input	Output for Sample Input
3 10 1 2 3 4 5 4 3 2 1 10 14 1 2 3 2 1 2 3 4 3 2 1 5 4 1 5 1 2 3 4 5	Case 1: 9 Case 2: 7 Case 3: 1

Note

Dataset is huge, use faster I/O methods.

SOLUTION

```
const int N = 100005 ;
int inp[N], a[N] , Lis[N] , Lds[N] , n , b[N];
void LIS()
{
    int i , tmp , low , lis = 0 ;
    a[0] = -INF ;
    a[n] = INF ;
    for ( i = 0 ; i < n ; i++ )
    {
        low = lower_bound( a , a+n+1 , inp[i] ) - a;
        a[low] = inp[i] ;
        lis = max ( lis , low );
        Lis[i] = lis ;
    }
}
void LDS()
{
    int i , tmp , low , lds = 0 ;
    b[0] = -INF ;
    b[n] = INF;
    for ( i = n-1 ; i >=0 ; i-- )
    {
        low = lower_bound( b , b+n+1 , inp[i] ) - b;
        b[low] = inp[i] ;
        lds = max ( lds , low );
        Lds[i] = lds ;
    }
}
int solve()
{
    int i , ans = 0 ;
    LIS();
    LDS();
    for ( i = 0 ; i < n ; i++ ) ans = max ( ans ,
min(Lis[i],Lds[i]));
    //printf("ans :: %d\n",ans) ;
    return ( 2 * ans );
}
```

```

void input()
{
    int i ;
    for ( i = 0 ; i < n ; i ++ ) scanf("%d",&inp[i]) , a[i] = b[i] =
INF;
}
int main()
{ int cs , t ;
  scanf("%d",&t);
  for( cs = 1 ; cs <= t ; cs++ )
  {   scanf("%d",&n);
      input();
      printf("Case %d: %d\n",cs,solve()-1);
  }
}

```

1422 - Halloween Costumes

Time Limit: 2 second(s)

Memory Limit: 32 MB

Gappu has a very busy weekend ahead of him. Because, next weekend is Halloween, and he is planning to attend as many parties as he can. Since it's Halloween, these parties are all costume parties, Gappu always selects his costumes in such a way that it blends with his friends, that is, when he is attending the party, arranged by his comic-book-fan friends, he will go with the costume of Superman, but when the party is arranged contest-buddies, he would go with the costume of 'Chinese Postman'.



Since he is going to attend a number of parties on the Halloween night, and wear costumes accordingly, he will be changing his costumes a number of times. So, to make things a little easier, he may put on costumes one over another (that is he may wear the uniform for the postman, over the superman costume). Before each party he can take off some of the costumes, or wear a new one. That is, if he is wearing the Postman uniform over the Superman costume, and wants to go to a party in Superman costume, he can take off the Postman uniform, or he can wear a new Superman uniform. But, keep in mind that, Gappu doesn't like to wear dresses without cleaning them first, so, after taking off the Postman uniform, he cannot use that again in the Halloween night, if he needs the Postman costume again, he will have to use a new one. He can take off any number of costumes, and if he takes off **k** of the costumes, that will be the last **k** ones (e.g. if he wears costume **A** before costume **B**, to take off **A**, first he has to remove **B**).

Given the parties and the costumes, find the minimum number of costumes Gappu will need in the Halloween night.

Input

Input starts with an integer **T** (≤ 200), denoting the number of test cases.

Each case starts with a line containing an integer **N** ($1 \leq N \leq 100$) denoting the number of parties. Next line contains **N** integers, where the i^{th} integer c_i ($1 \leq c_i \leq 100$) denotes the costume he will be wearing in party **i**. He will attend party 1 first, then party 2, and so on.

Output

For each case, print the case number and the minimum number of required costumes.

Sample Input	Output for Sample Input
2	Case 1: 3
4	Case 2: 4
1 2 1 2	

7	
1 2 1 1 3 2 1	

SOLUTION

```
const int N = 105 ;
int dp[N][N] , inp[N] , n;
int DP(int left , int right )
{
    int &ret = dp[left][right ];
    if(ret != -1 ) return ret;
    if(left > right ) return 0;
    ret = DP(left + 1 , right ) + 1;
    int i ;
    for ( i = left+1 ; i <= right ; i++ )
    {
        if( inp[left] == inp[i] ) ret = min ( ret , DP(left+1,i) +
DP(i+1,right ));
    }
    return ret;
}
int main()
{
    int cs, t , i ;
    scanf("%d",&t);
    for ( cs =1 ; cs<=t ; cs++ )
    {
        scanf("%d",&n);
        for ( i = 0 ; i < n ; i++ ) cin >> inp[i];
        memo(dp,-1);
        printf("Case %d: %d\n",cs,DP(0,n-1));
    }
    return 0;
}
```

1431 - The Party for the Rich

Time Limit: 8 second(s)

Memory Limit: 32 MB

A party has been arranged for n richest people of city Richtown. They have set some strange rules such that only rich people can join the party. The rule is that, when a persons enters the party, he is given two cards - **entrance** and **exit**. They are represented as (x, y) , where x is the integer written on the entrance card and y is the integer written on the exit card. No two persons can have entrance (or exit) cards with same integers written on them. When a person exits the party, he has to pay the dollar equivalent to the absolute difference of the integers in his entrance and exit cards.

Though they were rich, they were very clever. They planned that they will exchange their cards such a way that the total money paid by them is as low as possible. Any people can exchange cards multiple times and with multiple persons. But the entrance cards are distinguishable from the exit cards, so entrance cards are exchangeable with entrance cards only and same rule suffices for exit cards. For example, suppose there are three people having cards with $(1, 5)$, $(7, 3)$, $(8, 10)$, then if they don't exchange cards, they have to pay $|1-5| + |7-3| + |8-10| = 10\$$. But if they exchange them to $(1, 3)$, $(7, 5)$, $(8, 10)$ then they need to pay $|1-3| + |7-5| + |8-10| = 6\$$.

But there is one problem, each person must pay at least $K\$$, otherwise the organizers will suspect that they have been cheating. So, your job is to help them find the solution where they have to pay as less as possible without creating any suspicion.

Input

Input starts with an integer T (≤ 10), denoting the number of test cases.

Each case starts with a blank line. Next line contains two integers n ($1 \leq n \leq 10000$) and K ($0 \leq K \leq 2$). Each of the next n lines contains two integers (between 1 and 20000) denoting the integers written on the entrance and exit cards respectively for i^{th} person.

Output

For each case, print the case number and the minimum amount of money they need to pay. If its impossible to do so, print "**impossible**".

Sample Input	Output for Sample Input
2 3 1 1 1 7 3 8 10 1 2	Case 1: 10 Case 2: impossible

10 9	
------	--

Note

Dataset is huge, use faster I/O methods.

SOLUTION

Can't find any!